



Silk Test 20.0

Silk4J User Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
<http://www.microfocus.com>

© Copyright 1992-2019 Micro Focus or one of its affiliates.

MICRO FOCUS, the Micro Focus logo and Silk Test are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2019-05-02

Contents

Welcome to Silk4J 20.0	10
Licensing Information	11
Silk4J	12
Do I Need Administrator Privileges to Run Silk4J?	12
Best Practices for Using Silk4J	12
Automation Under Special Conditions (Missing Peripherals)	13
Silk Test Product Suite	14
What's New in Silk4J	16
Service Virtualization	16
Mobile Center Integration	16
Support for Universal Windows Platform Apps	16
Usability Enhancements	16
API Enhancements	17
Technology Updates	17
New Mozilla Firefox Versions	17
New Google Chrome Versions	17
Java 12 Support	17
New Microsoft Visual Studio Version	18
New Eclipse Version	18
New Java SWT Versions	18
Silk4J Quick Start Tutorial	19
Creating a Silk4J Project	19
Recording a Test for the Insurance Company Web Application	20
Replaying the Test for the Insurance Company Web Application	21
Working with Silk4J Projects	22
Creating a Silk4J Project	22
Importing a Silk4J Project	23
Creating Tests	24
Creating a Test for a Web Application	24
Creating a Test for a Standard Application	25
Creating a Test for a Mobile Web Application	25
Creating a Test for a Mobile Native Application	26
Recording a Test on Microsoft Edge	27
Recording a Test on Mozilla Firefox	28
Recording a Test on Google Chrome	29
Creating a Test Case Manually	30
Best Practices for Creating Test Scripts	30
Actions Available During Recording	31
Adding a Verification to a Script while Recording	32
Adding a Locator or an Object Map Item to a Test Method Using the Locator Spy	32
Including Custom Attributes in a Test	33
Characters Excluded from Recording and Replaying	34
Replaying Tests	35
Replaying Tests from Eclipse	35
Replaying a Test from the Command Line	35
Replaying Tests with Apache Ant	36
Troubleshooting when Replaying Tests with Ant	38
Replaying Tests from a Continuous Integration Server	38
Running Tests in Docker Containers	38
Silk Test Image Environment Variables	40

Example: Running Tests on Google Chrome	41
Example: Using docker-compose	42
Limitations when Running Tests in Docker Containers	43
Troubleshooting when Running Tests in Docker Containers	44
Replaying Silk4J Tests from Silk Central	44
Triggering Tests on Silk Central from a Continuous Integration Server	45
Replaying Tests in a Specific Order	45
Running Tests in Parallel	46
How Does Silk4J Synchronize Tests?	48
Enabling the Playback Status Dialog Box	49
Analyzing Test Results	51
Analyzing Test Results	51
HTML Reports	51
Visual Execution Logs with TrueLog	51
Enabling TrueLog	52
Changing the Location of the TrueLog	52
TrueLog Sections	52
Capturing the Contents of a Web Page	53
Why is TrueLog Not Displaying Non-ASCII Characters Correctly?	53
Silk Test Open Agent	54
Starting the Silk Test Open Agent	54
Stopping the Open Agent After Test Execution	54
Agent Options	54
Configuring the Connections Between the Silk4J Components	62
Configuring the Port to Connect to the Information Service	64
Configuring the Port to Connect to the Open Agent	65
Editing the Properties of the Silk Test Information Service	66
Replacing the Certificates that are Used for the HTTPS Connection to the Information Service	67
Remote Testing with the Open Agent	68
Testing with a Remote Open Agent	68
Configuring the Open Agent to Run Remotely in a NAT Environment	68
Base State	69
Modifying the Base State from the User Interface	69
Modifying the Base State in a Script	71
Running the Base State	72
Application Configuration	74
Modifying an Application Configuration	75
Select Application Dialog Box	76
Editing Remote Locations	77
Application Configuration Errors	78
Troubleshooting Application Configurations	78
Configuring Silk4J to Launch an Application that Uses the Java Network Launching Protocol (JNLP)	79
Creating a Test that Tests Multiple Applications	79
Setting Script Options	81
Setting TrueLog Options	81
Setting Recording Preferences	82
Setting Browser Recording Options	82
Setting Custom Attributes	84
Setting Classes to Ignore	85
Setting WPF Classes to Expose During Recording and Playback	85
Setting Synchronization Options	85
Setting Replay Options	86
Setting UI Automation Options	87

Setting Advanced Options	88
Setting Silk4J Preferences	89
Converting Projects to and from Silk4J	90
Converting a Java Project to a Silk4J Project	90
Converting a Silk4J Project to a Java Project	90
Testing Specific Environments	91
Active X/Visual Basic Applications	91
Dynamically Invoking ActiveX/Visual Basic Methods	91
Apache Flex Support	92
Configuring Flex Applications to Run in Adobe Flash Player	92
Launching the Component Explorer	93
Testing Apache Flex Applications	93
Testing Apache Flex Custom Controls	93
Customizing Apache Flex Scripts	103
Testing Multiple Flex Applications on the Same Web Page	103
Adobe AIR Support	104
Overview of the Flex Select Method Using Name or Index	104
Selecting an Item in the FlexDataGrid Control	105
Enabling Your Flex Application for Testing	105
Styles in Apache Flex Applications	116
Configuring Flex Applications for Adobe Flash Player Security Restrictions	117
Attributes for Apache Flex Applications	117
Why Cannot Silk4J Recognize Apache Flex Controls?	117
Java AWT/Swing Support	118
Attributes for Java AWT/Swing Applications	118
Dynamically Invoking Java Methods	118
Configuring Silk4J to Launch an Application that Uses the Java Network Launching Protocol (JNLP)	120
Determining the priorLabel in the Java AWT/Swing Technology Domain	120
Oracle Forms Support	120
Java SWT and Eclipse RCP Support	121
Java SWT Custom Attributes	122
Attributes for Java SWT Applications	122
Dynamically Invoking Java Methods	122
Troubleshooting Java SWT and Eclipse Applications	123
Testing Mobile Applications	124
Android	124
iOS	130
Recording Mobile Applications	143
Selecting the Mobile Device for Test Replay	143
Using Devices from Mobile Center Directly from Silk4J	144
Using Devices from Mobile Center through Silk Central	145
Installing the Certificate for an HTTPS Connection to Mobile Center	146
Changing the Mobile Center Password	147
Using SauceLabs Devices	147
Connection String for a Mobile Device	147
Interacting with a Mobile Device	151
Releasing a Mobile Device	151
Using the setLocation Method when Testing a Mobile Application	152
Troubleshooting when Testing Mobile Applications	152
Limitations for Testing Mobile Web Applications	158
Limitations for Testing Native Mobile Applications	159
Dynamically Invoking Methods for Native Mobile Apps	161
Clicking on Objects in a Mobile Website	162
Using Existing Mobile Web Tests	163

.NET Support	163
Windows Forms Support	163
Windows Presentation Foundation (WPF) Support	167
Silverlight Application Support	175
Visual COBOL Support	179
Rumba Support	179
Enabling and Disabling Rumba	180
Locator Attributes for Identifying Rumba Controls	180
Testing a Unix Display	180
SAP Support	181
Attributes for SAP Applications	181
Dynamically Invoking SAP Methods	181
Dynamically Invoking Methods on SAP Controls	182
Configuring Automation Security Settings for SAP	183
Universal Windows Platform Support	183
Troubleshooting when Testing UWP Apps	184
Windows API-Based Application Support	184
Attributes for Windows API-based Client/Server Applications	184
Determining the priorLabel in the Win32 Technology Domain	184
Testing Embedded Chrome Applications	185
Microsoft Foundation Class Support	186
Cross-Browser Testing	186
Selecting the Browser for Test Replay	187
Test Objects for xBrowser	189
Object Recognition for xBrowser Objects	189
Page Synchronization for xBrowser	190
Comparing API Playback and Native Playback for xBrowser	191
Setting Mouse Move Preferences	192
Browser Configuration Settings for xBrowser	192
Configuring the Locator Generator for xBrowser	194
Connection String for a Remote Desktop Browser	194
Testing Browsers on a Remote Windows Machine	195
Testing Google Chrome or Mozilla Firefox on a Mac	195
Setting Capabilities for WebDriver-Based Browsers	196
Testing with Apple Safari on a Mac	197
Testing with Google Chrome	201
Testing with Mozilla Firefox	205
Testing with Microsoft Edge	208
Responsive Web Design Testing	209
Detecting Visual Breakpoints	210
Improving iframe Performance	211
Testing Additional Browser Versions	213
Cross-Browser Testing: Frequently Asked Questions	214
Starting a Browser from a Script	218
Finding Hidden Input Fields	219
Attributes for Web Applications	219
Custom Attributes for Web Applications	219
Limitations for Testing on Microsoft Windows 8 and Microsoft Windows 8.1	220
Supported Attribute Types	220
Attributes for Apache Flex Applications	220
Attributes for Java AWT/Swing Applications	221
Attributes for Java SWT Applications	221
Attributes for SAP Applications	221
Locator Attributes for Identifying Silverlight Controls	222
Locator Attributes for Identifying Controls with UI Automation	223
Locator Attributes for Identifying Rumba Controls	224

Attributes for Web Applications	224
Attributes for Windows Forms Applications	224
Attributes for Windows Presentation Foundation (WPF) Applications	225
Attributes for Windows API-based Client/Server Applications	226
Dynamic Locator Attributes	226
Keyword-Driven Tests	228
Advantages of Keyword-Driven Testing	228
Keywords	229
Creating a Keyword-Driven Test in Silk4J	230
Recording a Keyword-Driven Test in Silk4J	230
Setting the Base State for a Keyword-Driven Test in Silk4J	232
Implementing a Keyword in Silk4J	232
Recording a Keyword in Silk4J	233
Marking a Test Method in a Script as a Keyword	234
Editing a Keyword-Driven Test	234
Managing Keywords in a Test in Silk Central	235
Which Keywords Does Silk4J Recommend?	237
Using Parameters with Keywords	238
Example: Keywords with Parameters	238
Combining Keywords into Keyword Sequences	240
Replaying Keyword-Driven Tests from Eclipse	240
Replaying Keyword-Driven Tests Which Are Stored in Silk Central	241
Replaying Keyword-Driven Tests from the Command Line	241
Replaying Keyword-Driven Tests with Apache Ant	242
Replaying a Keyword-Driven Test with Specific Variables	244
Integrating Silk4J with Silk Central	245
Implementing Silk Central Keywords in Silk4J	246
Uploading a Keyword Library to Silk Central	246
Uploading a Keyword Library to Silk Central from the Command Line	248
Searching for a Keyword	250
Filtering Keywords	250
Finding All References of a Keyword	250
Grouping Keywords	251
Troubleshooting for Keyword-Driven Testing	251
Object Recognition	252
Locator Basic Concepts	252
Object Type and Search Scope	252
Using Attributes to Identify an Object	253
Locator Syntax	253
Using Locators	255
Using Locators to Check if an Object Exists	256
Identifying Multiple Objects with One Locator	256
Locator Customization	257
Stable Identifiers	257
Custom Attributes	259
Troubleshooting Performance Issues for XPath	262
Locator Spy	262
Object Maps	264
Advantages of Using Object Maps	264
Turning Object Maps Off and On	265
Using Assets in Multiple Projects	265
Merging Object Maps During Action Recording	266
Using Object Maps with Web Applications	267
Renaming an Object Map Item	267
Modifying Object Maps	268

Modifying a Locator in an Object Map	269
Updating Object Maps from the Test Application	270
Copying an Object Map Item	271
Adding an Object Map Item	271
Opening an Object Map from a Script	272
Highlighting an Object Map Item in the Test Application	272
Finding Errors in an Object Map	273
Deleting an Object Map Item	273
Initially Filling Object Maps	274
Grouping Elements in Object Maps	274
Object Maps: Frequently Asked Questions	274
Can I Merge Multiple Object Maps Into a Single Map?	275
What Happens to an Object Map when I Delete a Test Script?	275
Can I Manually Create an Object Map for My Application Under Test?	275
Image Recognition Support	276
Image Click Recording	276
Image Recognition Methods	276
Image Assets	277
Creating an Image Asset	277
Adding Multiple Images to the Same Image Asset	278
Opening an Asset from a Script	279
Image Verifications	279
Creating an Image Verification	279
Adding an Image Verification During Recording	280
Using Assets in Multiple Projects	280
Enhancing Tests	282
Recording Additional Actions Into an Existing Test	282
Calling Windows DLLs	282
Calling a Windows DLL from Within a Script	282
DLL Function Declaration Syntax	283
DLL Calling Example	283
Passing Arguments to DLL Functions	284
Passing Arguments that Can Be Modified by the DLL Function	285
Passing String Arguments to DLL Functions	285
Aliasing a DLL Name	286
Conventions for Calling DLL Functions	286
Custom Controls	287
Dynamic Invoke	287
Adding Code to the Application Under Test to Test Custom Controls	288
Testing Apache Flex Custom Controls	291
Managing Custom Controls	291
Improving Object Recognition with Microsoft Accessibility	295
Using Accessibility	295
Enabling Accessibility	295
Overview of Silk4J Support of Unicode Content	296
Microsoft UI Automation	296
Recording a Test Against an Application with an Implemented UI Automation Provider Interface	297
Dynamically Invoking UI Automation Methods	298
Locator Attributes for Identifying Controls with UI Automation	300
Scrolling in UI Automation Controls	300
Limitations when Using UI Automation	301
Troubleshooting when Testing with UI Automation Support Enabled	302
Text Recognition Support	302
Service Virtualization	304
Discovering the Endpoints in your Application Under Test	304

Learning which Data is Transmitted Through an Endpoint	305
Simulating Data for Selected Endpoints	306
Simulating Data for Selected Endpoints through the API	306
Grouping Silk4J Tests	308
Why Do I Get the Error: Category cannot be resolved to a type?	310
Inserting a Result Comment in a Script	310
Consuming Parameters from Silk Central	310
Configuration Testing with Silk Central Connect	310
Measuring Execution Time	311
Slowing Down Tests	311
Testing Applications in Multiple UI Sessions on a Single Machine	311
Encrypting Passwords	312
Using Selenium WebDriver	314
Using Selenium with Existing Silk4J Scripts	314
Executing Selenium Scripts	314
Entering Special Keys Into A Text Field	316
Using Keyword-Driven Tests as Performance Tests	319
Known Issues	320
General Issues	320
Mobile Web Applications	322
Web Applications	322
Google Chrome	322
Internet Explorer	324
Microsoft Edge	325
Mozilla Firefox	325
SAP Applications	326
Silk4J	327
Enabling or Disabling Usage Data Collection	328
Contacting Micro Focus	329
Information Needed by Micro Focus SupportLine	329

Welcome to Silk4J 20.0



Welcome to Silk4J 20.0

[About Silk4J
Product Suite](#)



What's new

[Release Notes](#)



Featured sections

[Best Practices for Using Silk4J
Creating Tests
Testing Specific Environments](#)



Tutorials and demonstrations

[Quick Start Tutorial](#)



Online resources

[Micro Focus Home Page](#)
[Micro Focus Resource Page](#)
[Micro Focus Channel on YouTube](#)
[Online Documentation](#)
[Micro Focus SupportLine](#)
[Micro Focus Product Updates](#)
[Silk Test Knowledge Base](#)
[Silk Test Forum](#)
[Web-Based Training](#)



Provide feedback

[Contacting Micro Focus on page 329](#)
[Email us feedback regarding this Help](#)
[Suggest a feature](#)

Licensing Information

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



Note: A Silk Test license is bound to a specific version of Silk Test. For example, Silk Test 20.0 requires a Silk Test 20.0 license.

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

Licensing Mode	Application Type
Mobile Native	<ul style="list-style-type: none"> Mobile web applications. <ul style="list-style-type: none"> Android iOS Native mobile applications. <ul style="list-style-type: none"> Android iOS
Full	<ul style="list-style-type: none"> Web applications, including the following: <ul style="list-style-type: none"> Apache Flex Java-Applets Mobile web applications. <ul style="list-style-type: none"> Android iOS Apache Flex Java AWT/Swing, including Oracle Forms Java SWT and Eclipse RCP .NET, including Windows Forms and Windows Presentation Foundation (WPF) Rumba Windows API-Based <p> Note: To upgrade your license to a Full license, visit http://www.microfocus.com.</p>
Premium	<p>All application types that are supported with a <i>Full</i> license, plus SAP applications.</p> <p> Note: To upgrade your license to a Premium license, visit http://www.microfocus.com.</p>
Mobile Native Add-On	<p>In addition to the technologies supported with a Full or Premium license, the mobile native add-on license offers support for testing native mobile applications on Android and iOS.</p>

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.

Silk4J supports the testing of a broad set of application technologies.

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

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



Note: If you have opted not to display the start screen when you start Silk4J, you can check for available updates by clicking **Help > Check for Product Update**.



Note: To perform keyword-driven testing with Silk4J, your Eclipse platform is required to run on Java 7 or later.

Do I Need Administrator Privileges to Run Silk4J?

You require the following privileges to install or run Silk4J:

- To install Silk4J, you must have local administrator privileges.
- To install Silk4J on a Windows server, you must have domain-level administrator privileges.
- To run Silk4J, you require full access rights to the following folders, including all subfolders:
 - C:\ProgramData\Silk\SilkTest.
 - %APPDATA%\Roaming\Silk\SilkTest.
 - %APPDATA%\Local\Silk\SilkTest.
 - %TEMP%.

Best Practices for Using Silk4J

Depending on the application under test and the testing environment, you might face different challenges while trying to perform functional or regression tests against your application. Micro Focus recommends the following best practices:

- To optimally use the functionality that Silk4J provides, create an individual project for each application that you want to test, except when testing multiple applications in the same test.
- If you have a large test framework in place, consider using the keyword-driven testing approach.
- Allocate at least 512MB of memory for Eclipse. To do so, set the xms value in the file `eclipse.ini`, which is located in the Eclipse installation directory, to 512 or higher.
- Set the size of text and other items on the screen of the machine on which you are testing to the default value. When using another setting than the default, you might experience coordinate offset issues during record and replay.
 - If you are using Microsoft Windows 7, you can change this setting under **Start > Control Panel > Appearance and Personalization > Display**.
 - If you are using Microsoft Windows 10, you can change this setting under **Settings > System > Display > Change the size of text, apps and other items**.

- If you cannot record a test against your application because Silk4J cannot recognize the objects in the application or because Silk4J recognizes all objects in the application as `CONTROL`, you could try to use the Microsoft UI Automation support by enabling the **Enable UI Automation** option in Silk4J. For additional information, see *UI Automation*.

Automation Under Special Conditions (Missing Peripherals)

Basic product orientation

Silk4J is a GUI testing product that tries to act like a human user in order to achieve meaningful test results under automation conditions. A test performed by Silk4J should be as valuable as a test performed by a human user while executing much faster. This means that Silk4J requires a testing environment that is as similar as possible to the testing environment that a human user would require in order to perform the same test.

Physical peripherals

Manually testing the UI of a real application requires physical input and output devices like a keyboard, a mouse, and a display. Silk4J does not necessarily require physical input devices during test replay. What Silk4J requires is the ability of the operating system to perform keystrokes and mouse clicks. The Silk4J replay usually works as expected without any input devices connected. However, some device drivers might block the Silk4J replay mechanisms if the physical input device is not available.

The same applies to physical output devices. A physical display does not necessarily need to be connected, but a working video device driver must be installed and the operating system must be in a condition to render things to the screen. For example, rendering is not possible in screen saver mode or if a session is locked. If rendering is not possible, low-level replay will not work and high-level replay might also not work as expected, depend on the technology that is used in the application under test (AUT).

Virtual machines

Silk4J does not directly support virtualization vendors, but can operate with any type of virtualization solution as long as the virtual guest machine behaves like a physical machine. Standard peripherals are usually provided as virtual devices, regardless of which physical devices are used with the machine that runs the virtual machine.

Cloud instances

From an automation point of view, a cloud instance is not different to a virtual machine. However, a cloud instance might run some special video rendering optimization, which might lead to situations where screen rendering is temporarily turned off to save hardware resources. This might happen when the cloud instance detects that no client is actively viewing the display. In such a case, you could open a VNC window as a workaround.

Special cases

Application launched without any window (headless)

Such an application cannot be tested with Silk4J. Silk4J needs to hook to a target application process in order to interact with it. Hooking is not possible for processes that do not have a visible window. In such a case you can only run system commands.

Remote desktops,

If Silk4J resides and operates within a remote desktop session, it will fully operate as expected.

terminal services, and remote applications (all vendors)



Note: You require a full user session and the remote viewing window needs to be maximized. If the remote viewing window is not displayed for some reason, for example network issues, Silk4J will continue to replay but might produce unexpected results, depending on what remote viewing technology is used. For example, a lost remote desktop session will negatively impact video rendering, whereas other remote viewing solutions might show no impact at all once the viewing window was lost.

If Silk4J is used to interact with the remote desktop, remote view, or remote app window, only low-level techniques can be used, because Silk4J sees only a screenshot of the remote machine. For some remote viewing solutions even low-level operations may not be possible because of security restrictions. For example, it might not be possible to send keystrokes to a remote application window.

Known automation obstacles

Silk4J requires an interactively-logged-on full-user session. Disable anything that could lock the session, for example screen savers, hibernation, or sleep mode. If this is not possible because of organizational policies you could workaround such issues by adding *keep alive* actions, for example moving the mouse, in regular intervals or at the end of each test case.



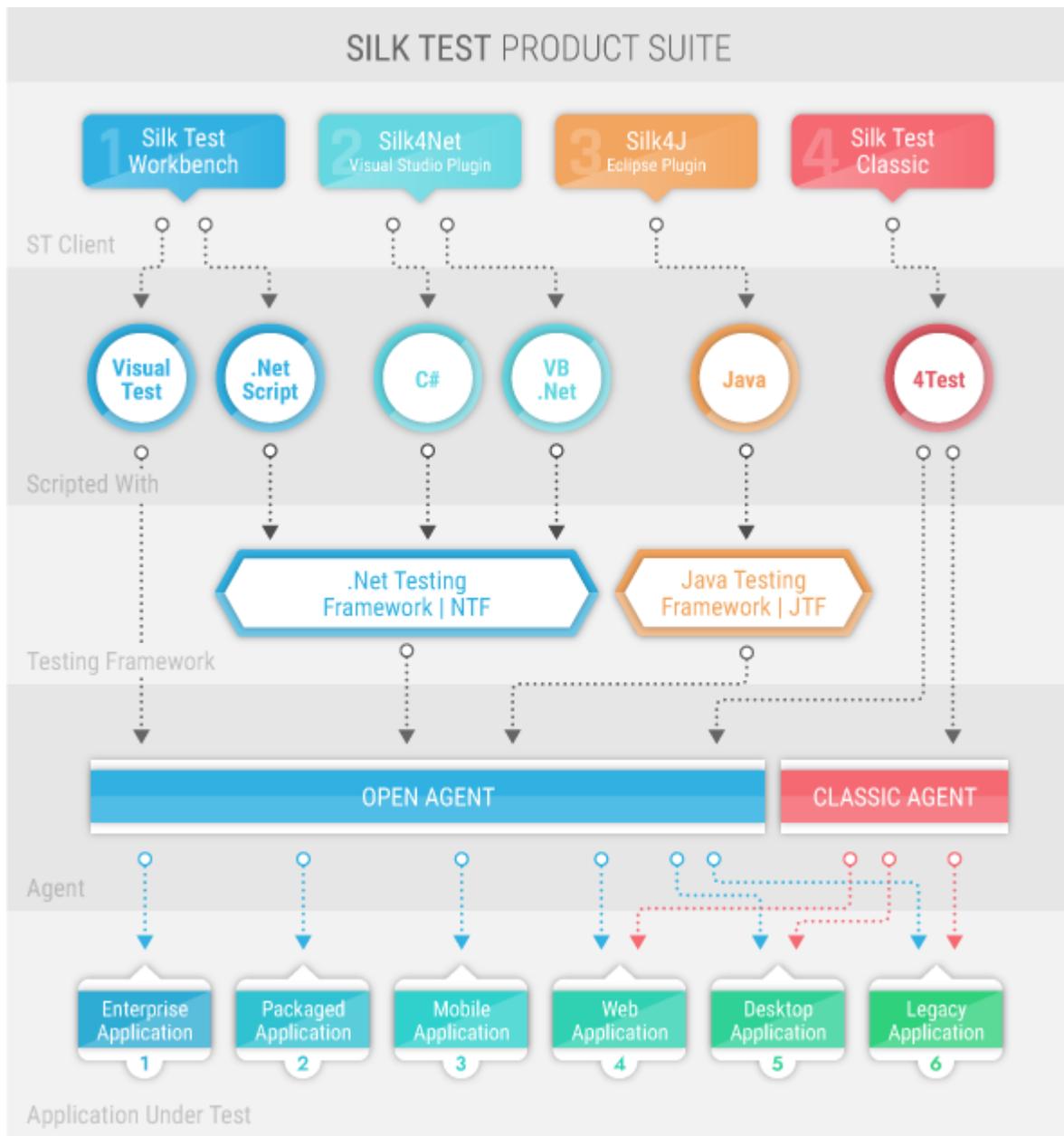
Note: Depending on the configuration of the actual testing environment and the technologies that are used for the AUT, the virtualization, and the terminal services, you may face additional challenges and limitations during the test automation process.

Silk Test Product Suite

Silk Test is an automated testing tool for fast and reliable functional and regression testing. Silk Test helps development teams, quality teams, and business analysts to deliver software faster, and with high quality. With Silk Test you can record and replay tests across multiple platforms and devices to ensure that your applications work exactly as intended.

The Silk Test product suite includes the following components:

- Silk Test Workbench – Silk Test Workbench is the quality testing environment that offers .NET scripting for power users and easy to use visual tests to make testing more accessible to a broader audience.
- Silk4NET – The Silk4NET Visual Studio plug-in enables you to create Visual Basic or C# test scripts directly in Visual Studio.
- Silk4J – The Silk4J Eclipse plug-in enables you to create Java-based test scripts directly in your Eclipse environment.
- Silk Test Classic – Silk Test Classic is the Silk Test client that enables you to create scripts based on 4Test.
- 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 sizes of the individual boxes in the image above differ for visualization purposes and do not reflect the included functionality.

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.

What's New in Silk4J

Silk4J supports the following new features:

Service Virtualization

Web applications typically consist of an HTML-based front-end and back-end services, which provide particular functionality like credit-card processing or user-management. The front-end heavily depends on those back-end services, and if the back-end is not accessible or under development, it is hard or even impossible to continue testing the front-end. With the new embedded SV capabilities of Silk4J, the back-end services are no longer a bottleneck for development testers. Instead, you can simply simulate key services and avoid waiting for the back-end to be ready. Using SV with Silk4J is as simple as recording a test case.

Mobile Center Integration

Silk4J now integrates directly with Mobile Center.

This integration allows you to:

- Leverage centrally managed mobile devices.
- Conveniently reserve mobile devices for testing.
- Effortlessly manage various versions of your mobile apps.
- Easily select the appropriate app during testing.
- Test your mobile apps across many platforms and devices.

Support for Universal Windows Platform Apps

Silk Test 20.0 supports UWP apps on the following operating systems:

- Microsoft Windows 10
- Microsoft Windows Server 2019

Usability Enhancements

This section lists usability enhancements that have been made in Silk Test 20.0.

Easily select mobile applications for testing

You can now easily select mobile applications from the UI, including the following:

- Mobile applications that are already installed on the selected mobile device.
- Mobile applications that are available in Mobile Center.
- Mobile applications from your network. Silk Test will install the selected application on the selected mobile device.

Encrypting passwords in the UI

Silk4J now supports encrypting passwords in the UI.

Support for <datalist> elements in web applications

Silk4J now supports testing <datalist> elements in web applications.

API Enhancements

This section lists API enhancements that have been made in Silk Test 20.0.

Wait until the value of a property is no longer equal to the specified value

You can now use the `waitForPropertyNotEquals` method to wait until the value of the specified property of a control no longer equals a specified value.

Wait until a control is visually stable

You can now use the `waitForScreenshotStable` method to wait until a control no longer visually changes and remains in the same position.

Technology Updates

This section lists the significant technology updates for Silk Test 20.0.

New Mozilla Firefox Versions

In addition to the versions of Mozilla Firefox, which have been tested with the previous version of Silk Test, Silk Test has now been tested with the following new versions of Mozilla Firefox:

- Mozilla Firefox 64
- Mozilla Firefox 65
- Mozilla Firefox 66



Note: This list includes the new versions of Mozilla Firefox that have been tested with Silk Test 20.0 until the release date of Silk Test 20.0. Silk Test 20.0 should be able to support newer versions of Mozilla Firefox, even if these versions have been released after the release date of Silk Test 20.0.

New Google Chrome Versions

In addition to the versions of Google Chrome, which have been tested with the previous version of Silk Test, Silk Test has now been tested with the following versions of Google Chrome:

- Google Chrome 71
- Google Chrome 72
- Google Chrome 73
- Google Chrome 74



Note: This list includes the versions of Google Chrome that have been tested with Silk Test 20.0 until the release date of Silk Test 20.0. Silk Test 20.0 should be able to test with newer versions of Google Chrome, even if these versions have been released after the release date of Silk Test 20.0.

Java 12 Support

Silk4J now supports testing applications that are based on Java 12.

New Microsoft Visual Studio Version

In addition to the editions of Microsoft Visual Studio, which were already tested with previous versions of Silk4NET, you can now integrate Silk4NET into the following editions of Microsoft Visual Studio:

- Microsoft Visual Studio 2019 Community.
- Microsoft Visual Studio 2019 Professional.
- Microsoft Visual Studio 2019 Enterprise.

New Eclipse Version

Silk4J now supports Eclipse 2019-03 (4.11).

New Java SWT Versions

Silk Test now supports testing standalone and Rich Client Platform (RCP) applications that are based on Java SWT 4.11.

Silk4J Quick Start Tutorial

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



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

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



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

Creating a Silk4J Project

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

1. In the Eclipse workspace, perform one of the following steps:

- Click the drop-down arrow next to the Silk Test toolbar icon  and choose **New Silk4J Project**.
- Right click in the **Package Explorer** and select **New > Other**. Expand the Silk4J folder and double-click **Silk4J Project**.
- If you installed or updated Silk4J to an existing Eclipse location, choose **File > New > Other**. Expand the Silk4J folder and double-click **Silk4J Project**.

The **New Silk4J Project** wizard opens.

2. In the **Project Name** text box, type a name for your project.

For example, type *Tutorial*.

3. *Optional:* Click **Next** to select the application that you want to test.

For the tutorial, click **Next**.

The **Select an application** page opens.

4. If you have not set an application configuration for the current project, select the tab that corresponds to the type of application that you are testing:

- If you are testing a standard application that does not run in a browser, select the **Windows** tab.
- If you are testing a web application or a mobile web application, select the **Web** tab.
- If you are testing a native mobile application, select the **Mobile** tab.

5. To test a standard application, if you have not set an application configuration for the current project, select the application from the list.

6. To test a web application or a mobile web application, if you have not set an application configuration for the current project, select one of the installed browsers or mobile browsers from the list.

- a) Specify the web page to open in the **Enter URL to navigate** text box. If an instance of the selected browser is already running, you can click **Use URL from running browser** to record against the URL currently displayed in the running browser instance. For the tutorial, select **Internet Explorer** and specify <http://demo.borland.com/InsuranceWebExtJS/> in the **Enter URL to navigate** text box.
- b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.

For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.

- c) *Optional*: Select an **Orientation** for the browser window.
 - d) *Optional*: Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.
7. To test a native mobile application (app) if you have not set an application configuration for the current project:
- a) Select the mobile device, on which you want to test the app, from the list.
 - b) Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.
 - If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
8. Click **Finish**. A new Silk4J project is created that includes the JRE system library and the required `.jar` files, `silktest-jtf-nodeps.jar` and `junit.jar`.
9. Select the type of test that you want to record:
- To bundle the recorded actions into one or more keywords, select **Keyword-Driven Test**.
 - To record the test without creating keywords, select **Silk Test JUnit Test**.
- For the tutorial, select **Silk Test JUnit Test**.
10. Click **Yes** to start recording a new Silk4J test or click **No** to return to the Eclipse workspace.
- For the tutorial, click **No**.

Recording a Test for the Insurance Company Web Application

Before you can create a Silk4J test, you must have created a Silk4J project.

Record a new test that navigates to the **Agent Lookup** page in the Insurance Company web application, <http://demo.borland.com/InsuranceWebExtJS/>. For a detailed version of how to record a test and how to configure test applications for each technology type, see the *Creating Tests* section of the Silk4J User Guide.

1. In the toolbar, click **Record Actions**.
2. Select the browser that you want to use.
3. Click **Record**. The application under test and the **Silk Recorder** open. Silk4J creates a base state and starts recording.
4. 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**.

For example, type 92121 as the zip code, `jsmith@gmail.com` as the email address and specify `Car` as the automobile type.
 - c) Specify an age, click a gender and driving record type, and then click **Next**.

For example, type 42 as the age, specify the gender as `Male` and `Good` as the driving record type.

- d) Specify a year, make, and model, click the financial info type, and then click **Next**.
For example, type 2010 as the year, specify Lexus and RX400 as the make and model, and Lease as the financial info type.
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 **Select Verification Type** dialog box opens.
- f) Select whether you want to create a verification of a property or an image verification.
For the tutorial, select **Verify properties of the TestObject**.
The **Verify Properties** dialog box opens.
- g) Check the **TextContents** check box and then click **OK**. A verification action is added to the script for the zip code text.
- h) Click **Home**.

An action that corresponds with each step is recorded.

5. Click **Stop**. The **Record Complete** dialog box opens.
6. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
7. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
8. In the **Test class** text box, specify the name for the test class.
For example, type: `AutoQuoteInput`.
To use an existing class, click **Select** and select the class that you want to use.
9. In the **Test method** text box, specify a name for the test method.
For example, type `autoQuote`.
10. Click **OK**.

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

Replaying the Test for the Insurance Company Web Application

1. Expand the **Tutorial** project in the Package Explorer.
2. Right-click the **AutoQuoteInput** class and choose **Run As > Silk4J Test** . If multiple browsers that are supported for replay are installed on the machine, the **Select Browser** dialog box opens.
3. Select the browser and click **Run**. When the test execution is complete, the **Playback Complete** dialog box opens.
4. Click **Explore Results** to review the TrueLog for the completed test. In this example, the verification will fail, because the **Zip Code** field in the test application is not cleaned.

Working with Silk4J Projects

This section describes how you can use Silk4J projects.

A Silk4J project contains all the resources needed to test the functionality of your applications by using Silk4J.

 **Note:** To optimally use the functionality that Silk4J provides, create an individual project for each application that you want to test, except when testing multiple applications in the same test.

Creating a Silk4J Project

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

1. In the Eclipse workspace, perform one of the following steps:

- Click the drop-down arrow next to the Silk Test toolbar icon  and choose **New Silk4J Project**.
- Right click in the **Package Explorer** and select **New > Other**. Expand the Silk4J folder and double-click **Silk4J Project**.
- If you installed or updated Silk4J to an existing Eclipse location, choose **File > New > Other**. Expand the Silk4J folder and double-click **Silk4J Project**.

The **New Silk4J Project** wizard opens.

2. In the **Project Name** text box, type a name for your project.

For example, type *Tutorial*.

3. *Optional:* Click **Next** to select the application that you want to test.

For the tutorial, click **Next**.

The **Select an application** page opens.

4. If you have not set an application configuration for the current project, select the tab that corresponds to the type of application that you are testing:

- If you are testing a standard application that does not run in a browser, select the **Windows** tab.
- If you are testing a web application or a mobile web application, select the **Web** tab.
- If you are testing a native mobile application, select the **Mobile** tab.

5. To test a standard application, if you have not set an application configuration for the current project, select the application from the list.

6. To test a web application or a mobile web application, if you have not set an application configuration for the current project, select one of the installed browsers or mobile browsers from the list.

- a) Specify the web page to open in the **Enter URL to navigate** text box. If an instance of the selected browser is already running, you can click **Use URL from running browser** to record against the URL currently displayed in the running browser instance. For the tutorial, select **Internet Explorer** and specify <http://demo.borland.com/InsuranceWebExt.JS/> in the **Enter URL to navigate** text box.
- b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
- c) *Optional:* Select an **Orientation** for the browser window.
- d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.

7. To test a native mobile application (app) if you have not set an application configuration for the current project:
 - a) Select the mobile device, on which you want to test the app, from the list.
 - b) Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.
 - If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
8. Click **Finish**. A new Silk4J project is created that includes the JRE system library and the required `.jar` files, `silktest-jtf-nodeps.jar` and `junit.jar`.
9. Select the type of test that you want to record:
 - To bundle the recorded actions into one or more keywords, select **Keyword-Driven Test**.
 - To record the test without creating keywords, select **Silk Test JUnit Test**.For the tutorial, select **Silk Test JUnit Test**.
10. Click **Yes** to start recording a new Silk4J test or click **No** to return to the Eclipse workspace.
For the tutorial, click **No**.

Importing a Silk4J Project

If you need to access Silk4J projects in a central repository, or from another machine, you can import the projects into your Eclipse workspace.

1. In Eclipse, create a new workspace. For additional information, refer to the Eclipse documentation.
2. In the Eclipse menu, click **File > Import**. The **Import** dialog box opens.
3. In the tree, expand the **General** node.
4. Select **Existing Projects into Workspace**.
5. Click **Next**. The **Import Projects** dialog box opens.
6. Click **Select root directory**.
7. Click **Browse** to browse to the location of the project.
8. Click **OK** in the **Browse For Folder** dialog box.
9. In the **Projects** list box, check the projects that you want to import.
10. In the **Import Projects** dialog box, Click **Finish**.

The selected projects are imported into the Eclipse workspace.

Creating Tests

Use Silk4J to create a test that uses XPath queries to find and identify objects. Typically, you use the **New Silk4J Test** wizard to create a test. After you create the initial test method, you can add additional test methods to an existing test class.

When you create a test, Silk4J automatically creates a base state for the application. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. For additional information, see *Base State*.

Creating a Test for a Web Application

Before you can create a Silk4J test, you must have created a Silk4J project.

To record a test for a web application:

1. In the **Package Explorer**, select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. Select the browser that you want to use.
4. *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
5. *Optional:* Select an **Orientation** for the browser window.
6. Click **Record**. The application under test and the **Silk Recorder** open. Silk4J creates a base state and starts recording.
7. *Optional:* To record WebDriver locators instead of recording Silk4J locators, click **WebDriver** in the **Silk Recorder**.

This feature is available when recording against one of the following browsers:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Apple Safari

For additional information, see *Using Selenium WebDriver*.

8. In the application under test, perform the actions that you want to test.
For information about the actions available during recording, see *Actions Available During Recording*.
9. Click **Stop**. The **Record Complete** dialog box opens.
10. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
11. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
12. In the **Test class** text box, specify the name for the test class.
For example, type: `AutoQuoteInput`.
To use an existing class, click **Select** and select the class that you want to use.
13. In the **Test method** text box, specify a name for the test method.
For example, type `autoQuote`.

14. Click **OK**.

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

Creating a Test for a Standard Application

Before you can create a Silk4J test, you must have created a Silk4J project.

To record a test for a standard application:

1. In the **Package Explorer**, select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the application under test, perform the actions that you want to test.
For example, you can choose menu commands such as **File > New** to test menu the menu command in your application. For information about the actions available during recording, see *Actions Available During Recording*.
4. Click **Stop**. The **Record Complete** dialog box opens.
5. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
6. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
7. In the **Test class** text box, specify the name for the test class.
8. In the **Test method** text box, specify a name for the test method.
9. Click **OK**.

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

Creating a Test for a Mobile Web Application

Before you can create a Silk4J test, you must have created a Silk4J project.

To record a new test for a mobile web application on a mobile device:

1. In the **Package Explorer**, select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the **Select Browser** dialog box, select the browser on the mobile device.
4. Click **Record**.
5. The **Recording** window opens and displays the screen of the mobile device. In the screen, perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* To record a swipe or a gesture, click and drag the mouse cursor.
 - d) *Optional:* If the action has parameters, type the parameters into the parameter fields in the **Choose Action** dialog box.
Silk4J automatically validates the parameters.
 - e) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

For additional information, see *Interacting with a Mobile Device*.

6. Click **Stop**. The **Record Complete** dialog box opens.
7. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
8. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
9. In the **Test class** text box, specify the name for the test class.
To use an existing class, click **Select** and select the class that you want to use.
10. In the **Test method** text box, specify a name for the test method.
11. Click **OK**.

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

Creating a Test for a Mobile Native Application

Before you can create a Silk4J test, you must have created a Silk4J project.

To record a new test for a mobile native application (app) on a mobile device:

1. In the **Package Explorer**, select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the **Select Mobile Device** dialog box, perform the following actions:
 - a) Select the mobile device, on which you want to test the app, from the list.
 - b) Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.
 - If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
4. Click **Record**.
5. The **Recording** window opens and displays the screen of the mobile device. In the screen, perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* To record a swipe or a gesture, click and drag the mouse cursor.
 - d) *Optional:* If the action has parameters, type the parameters into the parameter fields in the **Choose Action** dialog box.
Silk4J automatically validates the parameters.
 - e) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

For additional information, see *Interacting with a Mobile Device*.
6. Click **Stop**. The **Record Complete** dialog box opens.
7. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
8. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.

To use an existing package, click **Select** and select the package that you want to use.

9. In the **Test class** text box, specify the name for the test class.

To use an existing class, click **Select** and select the class that you want to use.

10. In the **Test method** text box, specify a name for the test method.

11. Click **OK**.

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

Recording a Test on Microsoft Edge

Before you can record a Silk4J test, you must have created a Silk4J project.

When starting the interaction with a web application on Microsoft Edge, Silk4J closes any open instance of Microsoft Edge and starts a new browser. This new browser uses a temporary profile without add-ons and with an empty cache. This instance of Microsoft Edge is closed when shutting down the Open Agent or when starting to test another application outside Microsoft Edge.



Note: You can currently not record keyword-driven tests on Microsoft Edge.

To record a new test for a web application on Microsoft Edge:

1. Select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the **Select Browser** dialog box, select the browser that you want to use.
4. *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
5. *Optional:* Select an **Orientation** for the browser window.
6. *Optional:* To record WebDriver locators instead of recording Silk4J locators, click **WebDriver** in the **Silk Recorder**.

This feature is available when recording against one of the following browsers:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Apple Safari

For additional information, see *Using Selenium WebDriver*.

7. Click **Record**.
8. The **Interactive Recording** window opens and displays the application under test. Perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text or specify parameters, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* If the action has parameters, type the parameters into the parameter fields.
Silk4J automatically validates the parameters.
 - d) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

During recording, Silk4J displays the mouse position next to the recording window. You can toggle the location to switch between displaying the absolute mouse position on the device display and the mouse position in relation to the active object. For additional information about the actions available during recording, see *Actions Available During Recording*.

9. Click **Stop**. The **Record Complete** dialog box opens.
10. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
11. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
12. In the **Test class** text box, specify the name for the test class.
To use an existing class, click **Select** and select the class that you want to use.
13. In the **Test method** text box, specify a name for the test method.
14. Click **OK**.

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

Recording a Test on Mozilla Firefox

Before you can record a Silk4J test, you must have created a Silk4J project.

To record a new test for a web application on Mozilla Firefox:

1. Select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the **Select Browser** dialog box, select the browser that you want to use.
4. *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
5. *Optional:* Select an **Orientation** for the browser window.
6. *Optional:* To record WebDriver locators instead of recording Silk4J locators, click **WebDriver** in the **Silk Recorder**.

This feature is available when recording against one of the following browsers:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Apple Safari

For additional information, see *Using Selenium WebDriver*.

7. Click **Record**.
8. The **Interactive Recording** window opens and displays the application under test. Perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text or specify parameters, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* If the action has parameters, type the parameters into the parameter fields.
Silk4J automatically validates the parameters.
 - d) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

During recording, Silk4J displays the mouse position next to the recording window. You can toggle the location to switch between displaying the absolute mouse position on the device display and the mouse position in relation to the active object. For additional information about the actions available during recording, see *Actions Available During Recording*.

9. Click **Stop**. The **Record Complete** dialog box opens.
10. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
11. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
12. In the **Test class** text box, specify the name for the test class.
To use an existing class, click **Select** and select the class that you want to use.
13. In the **Test method** text box, specify a name for the test method.
14. Click **OK**.

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

Recording a Test on Google Chrome

Silk4J supports recording a test on Google Chrome 50 or later. For previous versions of Google Chrome, Silk4J supports only replaying tests and recording locators.

Before you can record a Silk4J test, you must have created a Silk4J project.



Note: You cannot record tests on Google Chrome versions prior to version 50.

To record a new test for a web application on Google Chrome 50 or later:

1. Select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. In the **Select Browser** dialog box, select the browser that you want to use.
4. *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
5. *Optional:* Select an **Orientation** for the browser window.
6. *Optional:* To record WebDriver locators instead of recording Silk4J locators, click **WebDriver** in the **Silk Recorder**.

This feature is available when recording against one of the following browsers:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Apple Safari

For additional information, see *Using Selenium WebDriver*.

7. Click **Record**.
8. The **Interactive Recording** window opens and displays the application under test. Perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text or specify parameters, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* If the action has parameters, type the parameters into the parameter fields.
Silk4J automatically validates the parameters.
 - d) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

During recording, Silk4J displays the mouse position next to the recording window. You can toggle the location to switch between displaying the absolute mouse position on the device display and the mouse position in relation to the active object. For additional information about the actions available during recording, see *Actions Available During Recording*.

9. Click **Stop**. The **Record Complete** dialog box opens.
10. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.
11. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
12. In the **Test class** text box, specify the name for the test class.
To use an existing class, click **Select** and select the class that you want to use.
13. In the **Test method** text box, specify a name for the test method.
14. Click **OK**.

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

Creating a Test Case Manually

Typically, you use the **Base State** wizard to create a test case for Silk4J. Use this procedure if you want to manually create a test case.

1. Choose **File > New > JUnit Test Case**. The **New JUnit Test Case** dialog box opens.
2. Ensure the **New JUnit 4 test** option is selected. This option is selected by default.
3. In the **Package** text box, specify the package name.
By default, this text box lists the most recently used package. If you do not want to use the default package, choose one of the following:
 - If you have not created the package yet, type the package name into the text box.
 - If you have created the package already, click **Browse** to navigate to the package location and then select it.
4. In the **Name** text box, specify the name for the test case.
5. Click **Finish**. The new class file opens with code similar to the following:

```
package com.borland.demo;  
  
public class DynamicObjectRecognitionDemo {  
  
}
```

where `com.borland.demo` is the package that you specified and `DynamicObjectRecognitionDemo` is the class that you specified.

Connect to the test application by creating a base state or using an `attach` method.

Best Practices for Creating Test Scripts

The way in which you write your test cases might have a great impact on the performance and stability of your test set. During recording, Silk4J creates scripts that are as fast and stable as possible. However, there might be circumstances that require you to manually create or edit test scripts. This topic provides some general guidelines that might help you create test scripts that are maintainable, reusable, and lead to stable tests.

- Name your tests consistently and ensure that test names are self-explaining. Try to make the names correspond with the application under test and the tested functionality. For example, the test names *MyApp_SuccessfulLogin* and *MyApp_FailingLogin* are far easier to understand for other users than *Untitled_42* and *Untitled_43*.
- Describe your test cases as thoroughly as possible in a comment. Without a good description of the test case in natural language, someone who needs to change the implementing code might not be able to comprehend what exactly the test is doing.
- Ensure that your application under test is at the proper state when the test case starts. Return the application under test to the correct state before executing the actions in a test case.
- Ensure that your application under test is at a proper state when the test case finishes. If additional tests depend on the outcome of the test, ensure that they can start. Return the application under test to the correct state when the actions in a test case are executed.
- Whenever possible, ensure that your test cases are not depending on the results of other test cases. When this is not possible, ensure that the test cases are executed in the right order.
- Add verifications to your tests, to test the correctness of your application under test as well as the functional flow.
- Use keyword-driven testing to create highly reusable action sets. Bundle commonly used actions into keywords, combine keywords that are often executed sequentially into keyword sequences, and execute combinations of keywords and keyword sequences as keyword driven-tests.
- To keep your tests maintainable and reusable, prefer writing multiple simple test cases that are combinable to writing complex test cases.
- To avoid redundancies in your test set, prefer updating existing test cases to adding new test cases.

Actions Available During Recording

During recording, you can perform the following actions in the **Recording** window:

Action	Steps
Pause recording.	Click Pause to bring the AUT into a specific state without recording the actions, and then click Record to resume recording.
Change the sequence of the recorded actions.	To change the sequence of the recorded actions in the Recording window, select the actions that you want to move and drag them to the new location.
Select multiple actions.	To select multiple actions, press Ctrl and click on the actions or press Shift and click on the first and the last action that you want to select.
Replay recorded actions.	To replay recorded actions from the Recording window, select the actions and click Play . To select all recorded actions, click on Recorded Actions and then click Play .
Remove a recorded action.	To remove a falsely recorded action from the Recording window, hover the mouse cursor over the action and click Delete .
Verify an image or a property of a control.	Move the mouse cursor over the object that you want to verify and press Ctrl +Alt .
Change the object map entry	If you are recording against a web application or a mobile web app and if the automatically generated object map entry for a recorded object is difficult to read or contains special characters, you might want to change the object map entry to something more readable. You can do this during recording by right-clicking on the object and then expanding the Object identification area of the Choose Action dialog. Then you can edit the object map entry in the Object Map ID field. For example, the automatically generated object map entry for an image in our demo application is <i>http demo borland</i> . If you look at the object map, it is be difficult to understand what object this entry refers to. Changing the

Action	Steps
Select a different locator	<p>object map entry to something like <i>InsuranceWebHomePageBanner</i> would possibly provide more context. This functionality is available for all supported desktop and mobile browsers except Internet Explorer.</p> <p>If you are recording against a web application or a mobile web app and the automatically generated locator for a recorded object does not meet your requirements, you can click on the arrow in the Locator field and let Silk4J generate alternative locator suggestions for you. All suggested locators uniquely identify the object. This functionality is available for all supported desktop and mobile browsers except Internet Explorer.</p>

Adding a Verification to a Script while Recording

Do the following to add a verification to a script during recording:

1. Begin recording.
2. Move the mouse cursor over the object that you want to verify and press **Ctrl+Alt**.
When you are recording a mobile Web application, you can also click on the object and click **Add Verification**.
This option temporarily suspends recording and displays the **Select Verification Type** dialog box.
3. Select **Verify properties of the TestObject**.
For information about adding an image verification to a script, see *Adding an Image Verification During Recording*.
4. Click **OK**. The **Verify Properties** dialog box opens.
5. To select the property that you want to verify, check the corresponding check box.
6. Click **OK**. Silk4J adds the verification to the recorded script and you can continue recording.

Adding a Locator or an Object Map Item to a Test Method Using the Locator Spy

Manually capture a locator or an object map item using the **Locator Spy** and copy the locator or the object map item to the test method. For instance, you can identify the caption or the XPath locator string for GUI objects using the **Locator Spy**. Then, copy the relevant locator strings and attributes into the test methods in your scripts.

1. Open the test class that you want to modify.
2. In the Silk4J tool bar, click **Locator Spy**. The **Locator Spy** and the application under test open. If you are testing a mobile application, a recording window opens, representing the screen of the mobile device. You cannot perform actions in the recording window, but you can perform actions on the mobile device or emulator and then refresh the recording window.
3. *Optional:* To bring the application under test into the appropriate state before recording a locator, click **Stop Recording Locator**. The actions that you perform in the application under test are no longer recorded. To continue with the recording of a locator, click **Start Recording Locator**.
4. If you are testing a web application on Microsoft Edge, Mozilla Firefox, Google Chrome, or Apple Safari, select the **Recording Mode**:
 - Select **Record Silk Test Locators** to record locators that you can use with Silk4J.
 - Select **Record WebDriver Locators** to record locators that you can use with Selenium WebDriver.

 **Note:** When recording WebDriver locators, a ">" in the object tree denotes switching from one `IFrame` to another.

5. *Optional:* To display locators in the **Locator** column instead of object map items, uncheck the **Show object map identifiers** check box.

This setting is not available when recording WebDriver locators. Object map item names associate a logical name (an alias) with a control or a window, rather than the control or window's locator. By default, object map item names are displayed.

 **Note:** When you check or uncheck the check box, the change is not automatically reflected in the locator details. To update an entry in the **Locator Details** table, you have to click on the entry.

6. Position the mouse over the object that you want to record. The related locator string or object map item shows in the **Selected Locator** text box.

 **Note:** If you are testing on a browser, the **Selected Locator** field displays the locator only when you actually capture it.

7. Press **Ctrl+Alt** to capture the object.

 **Note:** Press **Ctrl+Shift** to capture the object if you specified the alternative record break key sequence on the **General Recording Options** page of the **Script Options** dialog box.

8. *Optional:* Click **Show additional locator attributes** to display any related attributes in the **Locator Attribute** table.

9. *Optional:* You can replace a recorded locator attribute with another locator attribute from the **Locator Attribute** table.

For example, your recorded locator might look like the following:

```
/BrowserApplication//BrowserWindow//input[@id='loginButton']
```

If you have a `textContent Login` listed in the **Locator Attribute** table, you can manually change the locator to the following:

```
/BrowserApplication//BrowserWindow//input[@textContent='Login']
```

The new locator displays in the **Selected Locator** text box.

10. If you are recording WebDriver locators, select the locator type for the **Selected Locator**:

- XPath locator. Identifies the control by combining the name of the control class and a collection of prioritized attributes into a unique locator. If the combined locator does not uniquely identify the control, Silk4J additionally adds an index to the locator or prefixes a parent UI control by using "//".
- Locate by id. Identifies the control by the id attribute.
- Locate by name. Identifies the control by the name attribute.
- Locate by link text. Only for hyperlinks.

11. To copy the locator, click **Copy Locator to Clipboard**.

In the **Selected Locator** text box, you can also mark the portion of the locator string that you want to copy, and then you can right-click the marked text and click **Copy**.

12. In the script, position your cursor to the location to which you want to paste the recorded locator.

For example, position your cursor in the appropriate parameter of a `Find` method in the script.

The test method, into which you want to paste the locator, must use a method that can take a locator as a parameter. Using the **Locator Spy** ensures that the locator is valid.

13. Copy the locator or the object map item to the test case or to the Clipboard.

14. Click **Close**.

Including Custom Attributes in a Test

You can include custom attributes in a test to make a test more stable. For example, in Java SWT, the developer implementing the GUI can define an attribute, such as `silkTestAutomationId`, for a widget

that uniquely identifies the widget in the application. A tester using Silk4J can then add that attribute to the list of custom attributes (in this case, `silkTestAutomationId`), and can identify controls by that unique ID.

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

 **Note:** You cannot set custom attributes for Flex or Windows API-based client/server (Win32) applications.

To include custom attributes in a test, include the custom attributes directly in the test that you create. For example, to find the first text box with the unique ID 'loginName' in your application, you can use the following query:

```
myWindow.find("./TextField[@silkTestAutomationId='loginName']")
```

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

For example in a Web application, to add an attribute called "bcauid" type:

```
<input type='button' bcauid='abc'  
value='click me' />
```

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

Characters Excluded from Recording and Replaying

The following characters are ignored by Silk Test during recording and replay:

Characters	Control
...	MenuItem
tab	MenuItem
&	All controls. The ampersand (&) is used as an accelerator and therefore not recorded.

Replaying Tests

This section provides detailed information on the various ways to replay your Silk4J tests.

For example, you can run Silk4J tests from Eclipse or by using the command line.

Replaying Tests from Eclipse

1. Navigate to the test method or keyword-driven test that you want to replay.
2. Perform one of the following steps:
 - Right-click a package name in the **Package Explorer** to replay all test methods or keyword-driven tests in the package.
 - 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 keyword-driven test name in the **Package Explorer** to replay the keyword-driven test.
 - 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** .



Note: Choosing **Run As > JUnit Test** is not supported as it will disable many Silk4J features.

4. If you are testing a web application, the **Select Browser** dialog box opens. Select the browser and click **Run**.
5. *Optional:* If necessary, you can press both **Shift** keys at the same time to stop the execution of the test.
6. 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 from the Command Line

You must update the PATH variable to reference your JDK location before performing this task. For additional information, see [JDK Installation for Microsoft Windows](#).

1. Include the following in the CLASSPATH:

- junit.jar.
- The org.hamcrest.core JAR file.
- silktest-jtf-nodeps.jar. This JAR is located in the ng\JTF subfolder of your Silk Test installation directory, for example C:\Program Files (x86)\Silk\SilkTest\ng\JTF.



Note: Micro Focus recommends using %OPEN_AGENT_HOME% instead of the full path to the ng directory, for example %OPEN_AGENT_HOME%\JTF, as this is more flexible and will even work if Silk Test is not installed in the default location.

- The JAR or folder that contains your compiled tests.
- If you are using service virtualization, add the SV Lab Client library.

```
%OPEN_AGENT_HOME%\SVLabConnector\SVLabClient\lib\*
```

The classpath should look similar to the following:

```
set CLASSPATH=<eclipse_install_directory>\plugins  
\org.junit4_4.3.1\junit.jar;<eclipse_install_directory>\plugins
```

```
\org.hamcrest.core_1.3.0.v201303031735.jar;%OPEN_AGENT_HOME%\JTF\silktest-jtf-nodeps.jar;C:\myTests.jar
```

If you are using service virtualization, the classpath should look similar to the following:

```
set CLASSPATH=<eclipse_install_directory>\plugins
\org.junit4_4.3.1\junit.jar;<eclipse_install_directory>\plugins
\org.hamcrest.core_1.3.0.v201303031735.jar;%OPEN_AGENT_HOME%\JTF\silktest-jtf-nodeps.jar;C:\myTests.jar;%OPEN_AGENT_HOME%\SVLabConnector\SVLabClient\lib\*;
```

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
```

Replaying Tests with Apache Ant

To perform the actions described in this topic, ensure that Apache Ant is installed on your machine.

To replay tests with Apache Ant, for example to generate HTML reports of the test runs, use the `SilkTestSuite` class. To replay keyword-driven tests with Apache Ant, use the `KeywordTestSuite` class. For additional information on replaying keyword-driven tests with Apache Ant, see *Replaying Keyword-Driven Tests with Apache Ant*.

1. To execute tests with Apache Ant, create a JUnit test suite with the `@SuiteClasses` annotation. For example, if you want to execute the tests in the classes `MyTestClass1` and `MyTestClass2`, which are located in the same Silk4J project, create the JUnit test suite `MyTestSuite` as follows:

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

2. Open the `build.xml` file of the Silk4J project, which includes the tests.

3. To execute the tests, add the following target to the `build.xml` file:



Note: The following code sample works only with Silk4J projects that are created with Silk Test 15.5 or later.

```
<target name="runTests" depends="compile">
  <condition property="agentRmiHost" value="">
    <not>
      <isset property="agentRmiHost" />
    </not>
  </condition>
```

```

<condition property="silkttest.configurationName" value="">
  <not>
    <isset property="silkttest.configurationName" />
  </not>
</condition>
<mkdir dir="./reports"/>
<junit printsummary="true" showoutput="true" fork="true">
  <sysproperty key="agentRmiHost" value="{agentRmiHost}" />
  <sysproperty key="silkttest.configurationName" value="{silkttest.configurationName}" />
  <classpath>
    <fileset dir="{output}">
      <include name="**/*.jar" />
    </fileset>
    <fileset dir="{buildlib}">
      <include name="**/*.jar" />
    </fileset>
  </classpath>

  <test name="MyTestSuite" todir="./reports"/>
</junit>
</target>

```

For additional information about the JUnit task, see <https://ant.apache.org/manual/Tasks/junit.html>.

4. *Optional:* To create XML reports for all tests, add the following code to the target:

```
<formatter type="xml" />
```

5. *Optional:* To create HTML reports out of the XML reports, add the following code to the target:

```

<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml" />
  </fileset>
  <report format="noframes" todir="./report/html" />
</junitreport>

```

For additional information about the JUnitReport task, see <https://ant.apache.org/manual/Tasks/junitreport.html>.

The complete target should now look like the following:

```

<target name="runTests" depends="compile">

  <mkdir dir="./reports"/>
  <junit printsummary="true" showoutput="true" fork="true">
    <sysproperty key="agentRmiHost" value="{agentRmiHost}" />
    <sysproperty key="silkttest.configurationName" value="{silkttest.configurationName}" />
    <classpath>
      <fileset dir="{output}">
        <include name="**/*.jar" />
      </fileset>
      <fileset dir="{buildlib}">
        <include name="**/*.jar" />
      </fileset>
    </classpath>

    <test name="MyTestSuite" todir="./reports"/>
  </junit>
  <junitreport todir="./reports">
    <fileset dir="./reports">
      <include name="TEST-*.xml" />
    </fileset>
    <report format="noframes" todir="./report/html" />
  </junitreport>

```

```
</junitreport>
</target>
```

6. To run the tests from Eclipse, perform the following actions:

- a) In the **Package Explorer**, right-click the `build.xml` file.
- b) Select **Run As > Ant Build ...**
- c) In the **Targets** tab of the **Edit Configuration** dialog box, check **runTests**.
- d) Click **Run**.

You can also execute the tests from the command line or from a CI server. For additional information, see <https://ant.apache.org/manual/running.html> and *Replaying Tests from a Continuous Integration Server* in the *Silk4J Help*.

Troubleshooting when Replaying Tests with Ant

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

- 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>
```

Replaying Tests from a Continuous Integration Server

To run Silk4J tests from a continuous integration (CI) server, a CI server needs to be configured. This topic uses Jenkins as an example.

1. Add a new job to the CI server to compile the Silk4J tests.
For additional information, refer to the documentation of the CI server.
2. Add a new job to the CI server to execute the Silk4J tests.
3. Replay the tests from the CI server by using an Apache Ant file. Running the tests with an Ant file creates JUnit results, while running the tests from the command line does not.

Whenever your CI job is executed, it also triggers the execution of the specified Silk4J tests. On Jenkins, the Ant output is displayed in the JUnit plug-in and the TrueLog file is saved.

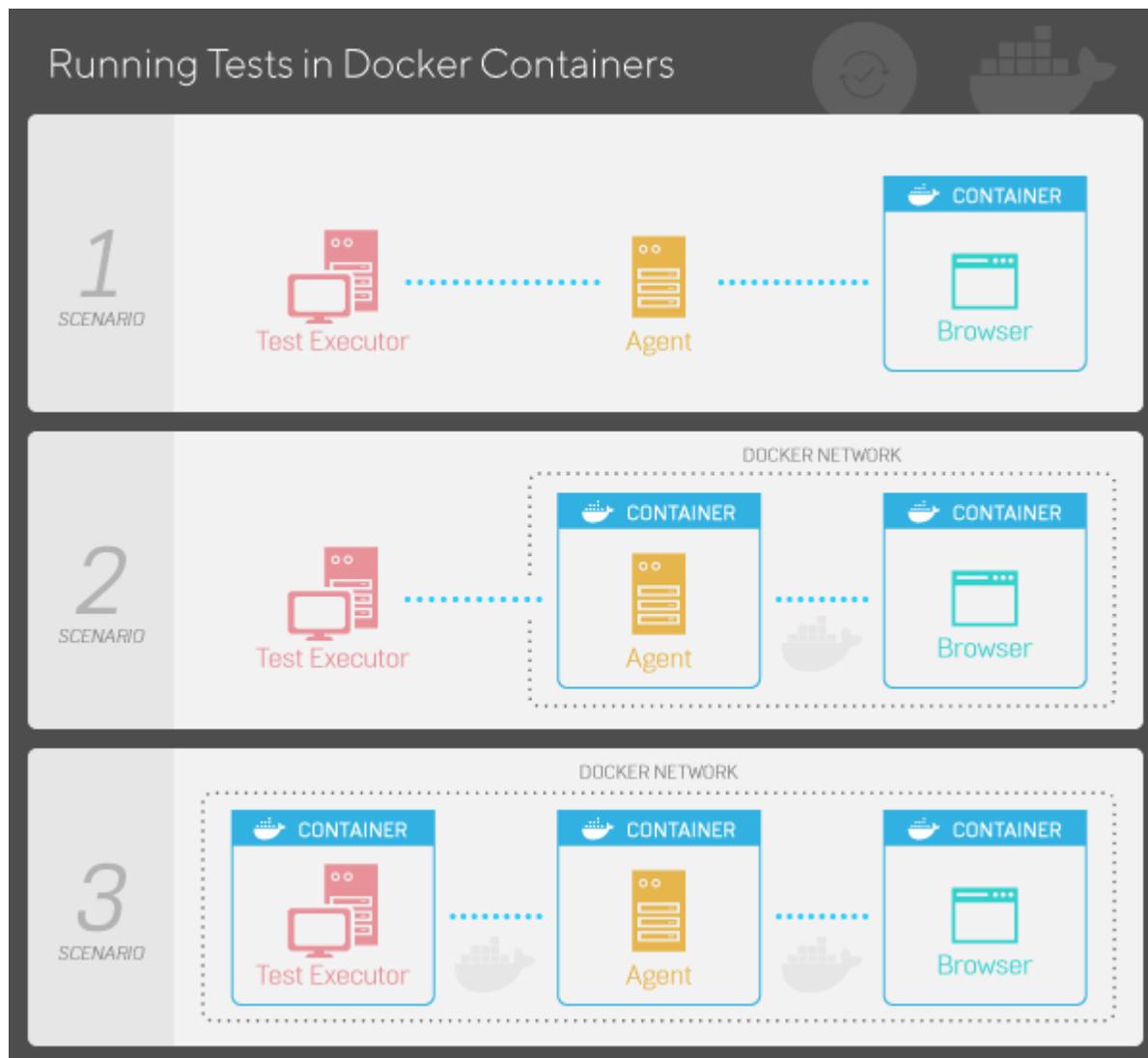
Running Tests in Docker Containers

Docker is a container platform provider that enables you to package applications, including all their dependencies, into virtual containers.

Silk4J supports the following scenarios for running tests from Docker containers:

- Running tests against a browser in a Docker container. The Open Agent and the test executor are running on your local Windows machine.

- Running tests against a browser in a Docker container through an Open Agent that is running in another Docker container. The browser and the Open Agent communicate with each other through a Docker network. The test executor is running on your local Windows machine.
- Running tests entirely from Docker containers. The Open Agent, the test executor, and the browser run in three separate Docker containers and communicate with each other through a Docker network.



Silk4J supports running tests for web applications in the following desktop browsers from Docker containers:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Apple Safari

Running your Silk4J tests in Docker containers enables you to better integrate your Silk4J tests into your existing CI workflow and provides the following advantages:

- You can run your tests on Windows, Linux, and macOS.
- You do not have to install Silk4J or any other required software, for example Java, on the machine on which you want to run the tests.

- You do not need to update the browser version when using the images provided by Docker.
- You can run tests in the background while working on other things.
- You can use the command line to run the Docker image and the tests.

Silk4J provides a basic Docker image, named *functionaltesting/silktest*, which contains the Open Agent and the capabilities to run Silk4J tests. You can download the image from the [Docker Hub](#).



Note: The topics in this section assume familiarity with Docker. If you are new to Docker and you would like to have additional information, refer to the [Docker](#) website.

Silk Test Image Environment Variables

The following table lists the environment variables that you can use to control how the Silk Test Docker container runs.

Environment Variable	Description
<code>SILK_LICENSE_SERVER</code>	The host name or IP address of the Silk Meter license server.
<code>SILK_SELENIUM_SERVER_PORT</code>	<p><i>Optional:</i> The listening port for the built-in Selenium server. If this environment variable is not set, the Selenium support is turned off.</p> <p>For example:</p> <pre>SILK_SELENIUM_SERVER_PORT=4444</pre>
<code>SILK_RMI_SERVER_PORT</code>	<p><i>Optional:</i> The listening port for the RMI server for the JTF. If this environment variable is not set, a dynamic port is calculated. Only set this if you need to interact with the Open Agent by using the JTF outside the Docker container and you require a fixed port for port forwarding.</p> <p>For example:</p> <pre>SILK_RMI_SERVER_PORT=30000</pre>
<code>SILK_LOG_FILE_PATH</code>	<p><i>Optional:</i> The path to the folder into which the test reports should be generated.</p> <p>For example:</p> <pre>SILK_LOG_FILE_PATH=/output</pre>
<code>SILK_CONSOLE_LOG</code>	<p><i>Optional:</i> Specifies whether to write the log to the console in addition to the Open Agent log file:</p> <p>For example:</p> <pre>SILK_CONSOLE_LOG=true</pre>
<code>SILK_LOG_LEVEL</code>	<p><i>Optional:</i> The log level.</p> <p>For example:</p> <pre>SILK_LOG_LEVEL=DEBUG</pre>

Prefix each variable with `-e`, for example:

```
docker run -e SILK_LOG_FILE_PATH=/logs
-v c:/temp/ChromeExample/logs:/logs
-e SILK_SELENIUM_SERVER_PORT=4444
-e SILK_RMI_SERVER_PORT=30000
-e SILK_CONSOLE_LOG=true
-e SILK_LOG_LEVEL=DEBUG
-e SILK_LICENSE_SERVER=<license-server address>
--name agent
functionaltesting/silktest:latest
```

Example: Running Tests on Google Chrome

This topic provides an example of how you can use Docker containers to run a Silk4J test set on Google Chrome on a Linux machine with Apache Ant. Before you can run the test set, you have to perform the following tasks:

- Install Docker on your machine.
- Prepare your Silk4J project for executing tests with Ant. For additional information, see [Replaying Tests with Apache Ant](#).
- Copy the project to your Linux machine, for example to `/home/<user name>/projects/InsuranceWeb`.

To run the Silk4J test set on Google Chrome.

1. Pull the latest version of the involved images to your registry.

For our example, we need the following three images:

- The latest Silk Test image.

```
docker pull functionaltesting/silktest:latest
```

- The latest Google Chrome image.

```
docker pull selenium/standalone-chrome:latest
```

- The Ant container that should run the tests.

```
docker pull webratio/ant:latest
```

2. Create a virtual Docker network to enable the Docker containers to communicate with each other.

For our example, name the network `my-network`.

```
docker network create my-network
```

3. Start the Google Chrome Docker container.

```
docker run --network my-network --name chrome selenium/standalone-chrome:latest
```

4. Start the Open Agent Docker container.

```
docker run -e SILK_LICENSE_SERVER=<license-server address>  
-e SILK_LOG_FILE_PATH=/logs  
-v /home/<user name>/projects/logs:/logs  
--network my-network  
--name agent  
functionaltesting/silktest:latest
```

For additional info on the available environment variables, see [Silk Test Image Environment Variables](#).

5. Start the Ant Docker container to run the tests.

```
docker run -v /home/<user name>/projects/InsuranceWeb:/tmp/project  
--network my-network  
--name=test-runner  
-it webratio/ant:1.10.1 ant  
-DagentRmiHost=agent:22902  
-Dsilktest.configurationName="host=http://chrome:4444/wd/  
hub;platformName=Linux - Chrome"  
-buildfile /tmp/project/build.xml runTests
```

6. View the test results under `/home/<user name>/projects/logs`.

7. *Optional:* To cleanup your test environment, execute the following commands:

- Stop the Open Agent Docker container.

```
docker stop agent
```

- Stop the Google Chrome Docker container.

```
docker stop chrome
```

- Remove the Open Agent Docker container.
`docker rm agent`
- Remove the Google Chrome Docker container.
`docker rm chrome`
- Remove the Ant Docker container.
`docker rm test-runner`
- Remove the virtual network.
`docker network rm my-network`

Example: Using docker-compose

This topic provides an example of how you can use Docker containers to run a Silk4J test set on Google Chrome on a Linux machine with Apache Ant. Before you can run the test set, you have to perform the following tasks:

- Install Docker on your machine.
- Run the "Hello, World" program from the command line to verify that your basic Docker setup is configured correctly:
`docker run hello-world`
- Prepare your Silk4J project for executing tests with Ant. For additional information, see [Replaying Tests with Apache Ant](#).
- Copy the project to your Linux machine, for example to `/home/<user name>/projects/InsuranceWeb`.

To run the tests in the Silk4J project on Google Chrome.

1. Create the docker-compose .yml file.

```
version: '3'
services:
  chrome:
    image: selenium/standalone-chrome:latest
    environment:
      - JAVA_OPTS=-Dwebdriver.Logger.level=WARNING
  agent:
    image: functionaltesting/silktest:latest
    environment:
      - SILK_LICENSE_SERVER=lnz-lic1.microfocus.com
      - SILK_LOG_FILE_PATH=/logs
    depends_on:
      - chrome
    links:
      - chrome
    volumes:
      - /home/<user name>/projects/logs:/logs
  tests-runner:
    image: webratio/ant:1.10.1
    volumes:
      - /home/<user name>/InsuranceWeb:/tmp/project
    command: ["ant", "-DagentRmiHost=agent:22902", "-Dsilktest.configurationName=host=http://chrome:4444/wd/hub;platformName=Linux - GoogleChrome", "-buildfile", "/tmp/project/build.xml", "runTests"]
    depends_on:
      - agent
    links:
      - agent
```



Note: In order for the `agentRmiHost` system property to work, your test script must create a `Desktop` object by calling the constructor with no arguments. Additionally, in order for the `silktest.configurationName` system property to work, your test script must create a `BrowserBaseState` object by calling the constructor with no arguments.

2. Pull the latest version of the involved images to your registry.

```
docker-compose pull
```

3. Execute the tests and stop all containers when the first container, in this case the Ant container, stops.

```
docker-compose up --abort-on-container-exit
```

4. View the test results under `/home/<user name>/projects/logs`.

5. *Optional:* Cleanup your test environment.

```
docker-compose down
```

You could combine the commands for pulling the latest versions, executing the tests, and cleaning up the test environment into a single command.

```
docker-compose pull && docker-compose up --abort-on-container-exit && docker-compose down
```

Limitations when Running Tests in Docker Containers

The following limitations apply when playing back tests in Docker containers:

- Image recognition, including `imageClick`, `imageRect`, `imageExists`, and image verifications, is not supported on the desktop. These methods are only supported for the `BrowserWindow` class and for the `BrowserApplication` class. For example, `desktop.<BrowserWindow>find("demo_borland_com.BrowserWindow").imageClick("LoginButton");` is supported, while `desktop.imageClick("LoginButton");` is not.
- The mouse methods and the keyboard input methods, for example `mouseMove`, `typeKeys`, and `click`, are supported for all objects except the `Desktop` object.
- The `getViewPortName` and `setViewPortName` methods of the `BrowserWindow` class are not supported.
- The following system functions are not supported:
 - `closeFile`
 - `closeIniFile`
 - `createRegistryKey`
 - `createRegistryValue`
 - `deleteRegistryKey`
 - `deleteRegistryValue`
 - `existsRegistryKey`
 - `fileWriteLine`
 - `getAgentVersion`
 - `getClipboardText`
 - `getCurrentDrive`
 - `getCursorPosition`
 - `getCursorType`
 - `getFileInfo`
 - `getFreeDiskSpace`
 - `getIniFileValue`
 - `getLocale`
 - `getRegistryKeyNames`
 - `getRegistryValue`
 - `getRegistryValueNames`

- `openFile`
- `openIniFile`
- `readLine`
- `setClipboardText`
- `setCurrentDrive`
- `setEnvironmentVariable`
- `setFilePointer`
- `setIniFileValue`
- `setRegistryValue`

Troubleshooting when Running Tests in Docker Containers

How can I see what a test running in a Docker container is currently doing?

To debug a test running against Google Chrome or Mozilla Firefox in a Docker container, you can use the debug images for Google Chrome or Mozilla Firefox that are available from the [Docker Hub](#). These will enable you to view the text execution through a VNC connection.

Replaying Silk4J Tests from Silk Central

To access Silk4J tests from Silk Central, you need to store the Silk4J tests in a JAR file in a repository that Silk Central can access through a source control profile.

To replay functional tests in Silk4J from Silk Central, for example keyword-driven tests:

1. In Silk Central, create a project from which the Silk4J tests will be executed.
2. Under **Tests > Details View**, create a new test container for the new project.

For additional information about Silk Central, refer to the [Silk Central Help](#).

The test container is required to specify the source control profile for the Silk4J tests.

- a) In the **Tests** tree, right-click on the node below which you want to add the new test container.
 - b) Click **New Test Container**. The **New Test Container** dialog box opens.
 - c) Type a name for the new test container into the **Name** field.
For example, type `Keyword-Driven Tests`
 - d) In the **Source control profile** field, select the source control profile in which the JAR file, which contains the Silk4J tests, is located.
 - e) Click **OK**.
3. Create a new JUnit test in the new test container.

For additional information about Silk Central, refer to the [Silk Central Help](#).

- a) In the **Test class** field of the **JUnit Test Properties** dialog box, type the name of the test class.
Specify the fully-qualified name of the test suite class. For additional information, see [Replaying Keyword-Driven Tests from the Command Line](#).
- b) In the **Classpath** field, specify the name of the JAR file that contains the tests.
- c) For keyword-driven testing, also specify the paths to the following files, separated by semicolons.
 - `com.borland.silk.keyworddriven.engine.jar`
 - `com.borland.silk.keyworddriven.jar`
 - `silktest-jtf-nodeps.jar`

These files are located in the Silk Test installation directory. For example, the **Classpath** field for the keyword-driven tests in the JAR file `tests.jar` might look like the following:

```
tests.jar;C:\Program Files
(x86)\Silk\SilkTest\ng\KeywordDrivenTesting
\com.borland.silk.keyworddriven.engine.jar;C:\Program Files
(x86)\Silk\SilkTest\ng\KeywordDrivenTesting
\com.borland.silk.keyworddriven.jar;C:\Program Files
(x86)\Silk\SilkTest\ng\JTF\silktest-jtf-nodeps.jar
```

4. Click **Finish**.
5. Execute the tests.

For additional information about executing tests in Silk Central, refer to the [Silk Central Help](#).

Triggering Tests on Silk Central from a Continuous Integration Server

To run Silk4J tests from a continuous integration server, the following infrastructure is required:

- A Silk Central server with the appropriate execution definitions.
-  **Note:** This topic focuses on the integration with Silk Central, but you could also use another test-scheduling tool.
- A continuous integration (CI) server, for example Hudson or Jenkins. This topic uses Jenkins as an example.

To replay functional tests from a CI server:

1. In Silk Central, retrieve the project ID and the execution plan ID of any execution plan that you want to run from the CI server.
 - a) Select **Execution Planning > Details View**.
 - b) In the **Execution Plans** tree, select the project that contains the execution. The **Project ID** is displayed in the **Properties** pane of the project.
 - c) In the **Execution Plans** tree, select the execution plan. The **Execution Plan ID** is displayed in the **Properties** pane of the execution plan.
2. Install the **SCTMExecutor** plugin on the CI server. This plugin connects the CI server to your Silk Central server.
3. Configure the **SCTMExecutor** plugin:
 - a) On Jenkins, navigate to the **Silk Central Test Manager Configuration** configuration in the global Jenkins configuration page.
 - b) Type the address of the Silk Central service into the **Service URL** field.
For example, if the server name is `sctm-server`, type `http://sctm-server:19120/services`.
4. Extend your CI build job.
 - a) On Jenkins, select **Silk Central Test Manager Execution** from the **Add build step** list.
 - b) Type the ID of the execution plan into the **Execution Plan ID** field.

You can execute an arbitrary number of execution plans by separating the IDs with a comma.
 - c) Type the project ID of the Silk Central project into the **SCTM Project ID** field.

Whenever your CI build job is executed, it also triggers the execution of the specified Silk Central execution plans.

Replaying Tests in a Specific Order

With Java 1.6 or prior, JUnit tests are executed in the order in which they are declared in the source file.



Note: With Java 1.7 or later, you cannot specify the order in which the JUnit tests are executed. This is a JUnit limitation for test execution.

JUnit tests are executed differently, depending on the JUnit version. With a JUnit version prior to 4.11 the tests are executed in no particular order, which may differ between test runs. With JUnit 4.11 or higher the tests are executed in the same order for each test run, but the order is unpredictable.

Depending on your testing environment you might be able to workaround this limitation.

Examples for a workaround

If your test set does not include modules and suites, you could add the following lines to the start of the source file:

```
import org.junit.FixMethodOrder;
import org.junit.runners.MethodSorters;
@FixMethodOrder(MethodSorters.JVM)
```

There are three possible values you can specify for the `FixMethodOrder`:

MethodSorters.JVM

The order in which the methods are returned by the JVM, potentially a different order for each test run. Might break your test set.

MethodSorters.DEFAULT

Deterministic ordering based upon the hashCode of the method name. Changing the order is difficult because you have to define method names that lead to an appropriate hashCode.

MethodSorters.NAME_ASCENDING

The order is based upon the lexicographic ordering of the names of the tests. You would have to rename your tests so that the alphabetical order of the test names matches the order in which you want the tests to be executed.

You could also use a Java version prior to 1.7.

Running Tests in Parallel

You can use multiple JUnit processes to execute tests in parallel against multiple browsers or mobile devices. For example, you can use this functionality when executing test from a continuous integration server, or from Silk Central.

Silk Testby default supports parallel testing for the following browsers and platforms:

- Google Chrome.
- Mozilla Firefox.
- Web, native, and hybrid apps on the following platforms:

- Physical Android devices.
- Android Emulators.
- Physical iOS devices.

When using mixed scripts to add WebDriver functionality to an existing Silk4J script, use the `getWebDriver` method to get a new driver. For additional information, see [Using Selenium with Existing Silk4J Scripts](#).

 **Note:** Using new `RemoteWebDriver` to get a new WebDriver object does not work for parallel testing of mixed scripts.

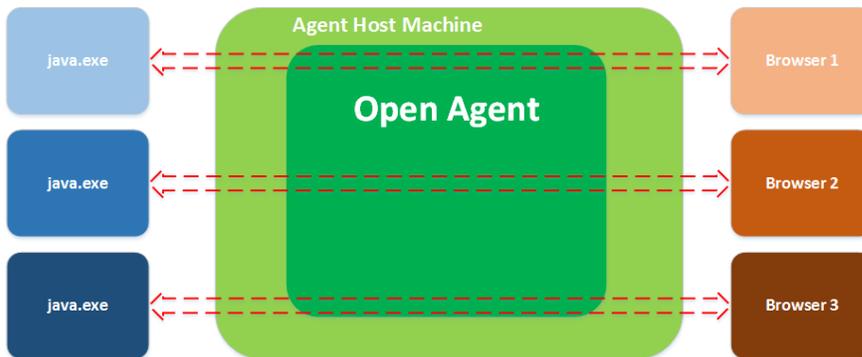
To disable parallel test replay, set the environment variable `SILKTEST_ENABLE_PARALLEL_TESTING` to false.

 **Note:** Enabling parallel testing causes the Open Agent to handle each test-executing process separately. Applications which have been tested in one Silk Test client cannot be tested from another client, while the initial client is running. For example, you cannot test the same application alternating between Silk4J and Silk4NET.

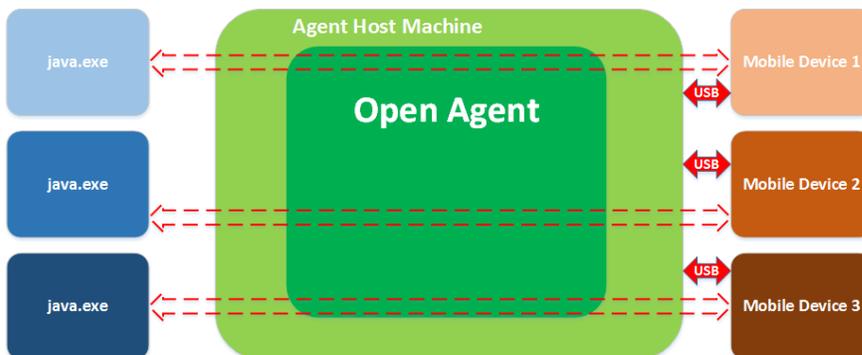
 **Note:** You cannot execute multiple test runs on the same mobile device at the same time. Before running tests in parallel, ensure that enough devices or emulators are available. Any test runs that get no mobile device or emulator assigned will fail.

Each parallel test run starts as a separate `java.exe`, which corresponds to one browser or mobile device. You can specify the browser or mobile device that you want to associate with a specific `java.exe` through the connection string. For additional information, see [Connection String for a Mobile Device](#) or [Connection String for a Remote Desktop Browser](#).

The following image shows testing multiple browsers in parallel:



The following image shows testing multiple devices in parallel:



Multiple processes starting simultaneously might each try to start the Open Agent on the machine on which Silk4J is running. Running the Open Agent multiple times on the same machine is not possible and will cause Silk4J to throw an exception. To avoid this, ensure that the Open Agent is running before starting the parallel test runs.

The test results are stored in multiple TrueLog files, one for each test run. To ensure that the TrueLog files are not overwritten, you can add placeholders to the TrueLog file name. For additional information, see *Setting TrueLog Options*.



Note: If you are experiencing high memory consumption during testing, ensure that test results are saved in the compressed TLZ file format, and not in the XLG format. Silk4J does not support the TrueLog API for parallel testing.

How Does Silk4J Synchronize Tests?

Many unexpected test failures are related to synchronization issues. Weak synchronization during test replay might generate false negative results, making it difficult to detect actual application problems. Synchronization errors are timing issues that heavily depend on the test environment, making these errors difficult to reproduce and solve. For example, a synchronization error that occurs in a specific test environment might never be reproduced in the development environment. Weak synchronization is one of the most common reasons for an automation project to get unmanageable with a growing set of automated tests.

Silk4J provides automated *test synchronization* for all supported technologies, enabling you to build robust and manageable test sets. During the replay of a test, Silk4J ensures that the test always waits until the AUT is ready to perform the next action. For a verification step in a test, Silk4J ensures that any preceding actions in the test are completed before performing the verification.

To adapt your tests to the specific behavior of your AUT, you can change the values of the following synchronization timeouts:

Synchronization timeout (OPT_SYNC_TIMEOUT)

The maximum time in milliseconds that Silk4J waits for the AUT to become ready during playback. The default value is 300000 milliseconds.

Object resolve timeout (OBJ_WAIT_RESOLVE_OBJDEF)

The maximum time in milliseconds that the `find` method searches for an object. The default value is 5000 milliseconds.



Note: To be able to successfully run tests under load or on slow systems, for example a virtual machine accessed through a slow connection, Silk4J occasionally increases the internal timeout, for example while the AUT is starting and when a dialog or window appears. As soon as the AUT, dialog, or window is completely started, Silk4J reduces the timeout again to the value you have specified for `OPT_WAIT_RESOLVE_OBJDEF`.

Object resolve retry interval (OPT_WAIT_RESOLVE_OBJDEF_RETRY)

If Silk4J cannot immediately find an object, Silk4J will retry to find the object until the object resolve timeout expires. The object resolve retry interval specifies the time in milliseconds that Silk4J waits before retrying to find the object. The default value is 1000 milliseconds.

Object enabled timeout (OPT_OBJECT_ENABLED_TIMEOUT)

The maximum time in milliseconds that Silk4J waits for an object to become enabled during playback. The default value is 1000 milliseconds.



Note: The timeouts do not overlap.

For detailed information about the automated synchronization that Silk4J provides specifically for Web applications, see *Page Synchronization for xBrowser*. For detailed information about the synchronization

that Silk4J provides specifically for Ajax applications, see [How to Synchronize Test Automation Scripts for Ajax Applications](#).

In addition to the automated synchronization, Silk4J also enables you to manually add wait functions to your scripts. Silk4J provides the following wait functions for manual synchronization:

waitForObject	Waits for an object that matches the specified locator. Works with an XPath locator or an object map identifier.
waitForProperty	Waits until the specified property gets the specified value or until the timeout is reached.
waitForPropertyNotEquals	Waits until the value of the specified property no longer equals the specified value or until the timeout is reached.
waitForDisappearance	Waits until the object does not exist or until the timeout is reached.
waitForChildDisappearance	Waits until the child object specified by the <code>locator</code> parameter does not exist or until the timeout is reached.
waitForScreenshotStable	Waits until the object is visually stable and does not change its position.

If a test randomly fails with an `ObjectNotFoundException`, increase the `Object` resolve timeout, for example to 30 seconds. For very specific long running operations, for example a click on an object that displays after a long calculation with a progress dialog, manually add the `waitForObject` method to the test script, to wait until the object is found, or add the `waitForDisappearance` method to the test script, to wait until the progress dialog is no longer displayed.

Automated synchronization example

Consider the following code sample, where Silk4J tries to click on a button with the caption **Ok**:

```
PushButton button = _desktop.find("//PushButton[@caption='ok']");  
button.Click();
```

To replay the actions in this code sample, Silk4J performs the following synchronization actions:

1. Silk4J tries to find the button. If the `Object` resolve timeout runs out, Silk4J stops the replay and throws an exception.
2. Silk4J waits until the application under test (AUT) is ready. If the `Synchronization` timeout runs out before the AUT is ready, Silk4J stops the replay and throws an exception.
3. Silk4J waits until the button is enabled. If the `Object` enabled timeout runs out before the button is enabled, Silk4J stops the replay and throws an exception.
4. Silk4J clicks the button.
5. Silk4J waits until the application under test (AUT) is ready again.

Enabling the Playback Status Dialog Box

Enable the **Playback Status** dialog box to view the actions that are performed during the replay of a test. This dialog box is very useful when replaying tests on another machine, for example on a remote Mac or on a mobile device.

To enable the **Playback Status** dialog box:

1. Click **Silk4J > Edit Options**.
2. Click the **Replay** tab.

3. To enable the **Playback Status** dialog box, check the **OPT_SHOW_PLAYBACK_STATUS_DIALOG** check box.
4. To display a video or screenshots of the application under test in the **Playback Status** dialog box, check the **OPT_PLAYBACK_STATUS_DIALOG_SCREENSHOTS** check box.
5. Click **OK**.



Note: If you are testing on a remote Mac or on a mobile device, ensure that the Mac or device does not switch off the screen during testing, otherwise the **Playback Status** dialog box will not display anything.

Analyzing Test Results

To enable you to analyze your tests after executing them, for example to find out why and how a test has failed, Silk4J generates test reports during test execution.

By default, Silk4J writes both a TrueLog and an HTML report when running a test. You can select which result formats Silk4J should generate under **Silk4J > Edit Options > TrueLog > Select result format**.

Analyzing Test Results

After running a test, you can review the test results and analyze the success or failure of the test run.

1. Run a Silk4J test. When the execution is finished, the **Playback Complete** dialog box opens.
2. Click **Explore Results** to examine the results of the tests.
3. Click through the results.

Silk4J captures a screenshot whenever a test fails.

HTML Reports

By default, Silk4J creates an HTML report and a TrueLog report when running tests. Both report formats include summary information about the test run and detailed information about the executed actions.

When a test run is finished, you can access the HTML report from the **Playback Complete** dialog.

By default, the HTML report is created in a sub-directory of the working directory of the process that executed the Silk4J tests. You can change the location under **Silk4J > Edit Options > TrueLog > TrueLog location**.

Visual Execution Logs with TrueLog

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



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

For additional information about TrueLog Explorer, refer to the [Silk TrueLog Explorer Help for Silk Test](#).

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



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

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

Enabling TrueLog

To enable TrueLog:

1. Click **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **TrueLog** tab.
3. In the **Basic Settings** 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.

By default, the TrueLog file is created in the working directory of the process that executed the Silk4J tests. To specify a different location for the TrueLog, click **Silk4J > Edit Options** to open the **Script Options** dialog box and click **Browse** to the right of the **TrueLog file** field.

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.

Changing the Location of the TrueLog

By default, the TrueLog is created in the working directory of the process that executed the Silk4J tests.

To specify another location for the TrueLog:

1. In the menu, click **Silk4J > Edit Options**. The **Script Options** dialog box displays.
2. Select the **TrueLog** tab.
3. Click **Browse** to the right of the **TrueLog location** field.

When running a test in the current project, the TrueLog is saved to the specified location.

TrueLog Sections

You can add sections to TrueLogs to add structure to complex scripts and to logically divide scripts into smaller named parts.

TrueLog sections can be nested and the same name can be used for multiple sections, which means the names do not have to be unique.

To create a new section in a TrueLog file, use the following code:

```
openTrueLogSection("section_name")
```

To close the last TrueLog section that was opened, use the following code:

```
closeCurrentTrueLogSection()
```

Sections are automatically closed in the following cases:

- The test case ends.
- The test class ends.
- The test run ends.
- A keyword ends.
- An exception occurs.

The following code sample demonstrates the usage of TrueLog sections:

```
@Test
public void subSections() {
    Desktop desktop = new Desktop();
    desktop.openTrueLogSection("Section a");
}
```

```
desktop.logInfo("In section a");
desktop.openTrueLogSection("Section b");
desktop.logInfo("In section b");
desktop.closeCurrentTrueLogSection();
desktop.logInfo("In section a again");
}
```

For additional information on the supported methods, refer to the API documentation of your Silk Test client.

Capturing the Contents of a Web Page

To capture a screenshot of the the part of the web page that is currently visible in the browser window, you can use the `captureBitmap` method. You have to specify the absolute or relative file path to the location and the name for the image file as a parameter. For example:

```
browserWindow.captureBitmap("C:\Temp\MyPage.png");
```

To capture a screenshot of the entire contents of a web page as a single image, you can use the `captureFullPageBitmap` method. You have to specify the absolute or relative file path to the location and the name for the image file as a parameter. For example:

```
browserWindow.captureFullPageBitmap("C:\Temp\MyPage.png");
```

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.

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 (in Microsoft Windows 7) **Start > Programs > Silk > Silk Test > Tools > Silk Test Open Agent** or (in Microsoft Windows 10) **Start > Silk > Silk Test Open Agent**. The Silk Test Open Agent icon  displays in the system tray.

Stopping the Open Agent After Test Execution

You can stop the Open Agent by using the command-line option `-shutDown` or from a script, to ensure that the agent does not continue running after the end of the test execution. The following code sample shows how you can stop the Open Agent from the command line:

```
openAgent.exe -shutDown
```

To stop the agent from a script:

1. Open or create a script that is executed when the test execution is finished.
For example, open an existing script that is used for cleanup after test execution.
2. Add the `shutDown` method to the script.



Note: The Open Agent will restart as soon as the agent is required by another script.

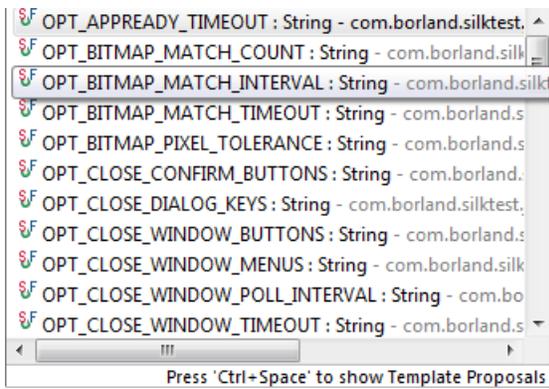
Agent Options

This section describes the options that can be manipulated with the `getOption` and `setOption` methods.



Note: By default, you can set global options by choosing **Silk4J > Edit Options** in the menu.

To set or get agent options in a script, use the `getOption` or `setOption` methods. For instance, when you type, `Desktop.setOption(Commonoptions.`, the agent options display automatically in the active editor window with the auto-completion and syntactical assistance technology. For instance, you might see the following options in the active editor window:



An example for inserting an option into a script would be the following:

```
desktop.setOption(Commonoptions.ApplicationReadyTimeout, 100000);
```

Alternatively, you can use the agent option name in scripts. For instance, you can type:

```
desktop.setOption("OPT_APPREADY_TIMEOUT", 100000);
```

.NET Option Name	Agent Option Name	Constant Type	Description
ApplicationReadyTime out	OPT_APPREADY_TIMEOUT	NUMBER	Specifies the number of milliseconds to wait for a newly launched application to become ready. If the application is not ready within the specified timeout, Silk4J raises an exception.
BitmapMatchCount	OPT_BITMAP_MATCH_COUNT	INTEGER	Specifies the number of consecutive snapshots that must be the same for the bitmap to be considered stable. Snapshots are taken up to the number of seconds specified by OPT_BITMAP_MATCH_TIMEOUT, with a pause specified by OPT_BITMAP_MATCH_INTERVAL occurring between each snapshot. By default, this is 0.
BitmapMatchInterval	OPT_BITMAP_MATCH_INTERVAL	REAL	Specifies the time interval between snapshots to use for ensuring the stability of the bitmap image. The snapshots are taken up to the time specified by OPT_BITMAP_MATCH_TIMEOUT. By default, this is 0.1.
BitmapMatchTimeout	OPT_BITMAP_MATCH_TIMEOUT	REAL	Specifies the total time allowed for a bitmap image to become stable. During the time period, Silk4J takes multiple snapshots of the image, waiting the number of seconds specified with OPT_BITMAP_MATCH_TIMEOUT between snapshots. If the value returned by

.NET Option Name	Agent Option Name	Constant Type	Description
			<p>OPT_BITMAP_MATCH_TIMEOUT is reached before the number of bitmaps specified by OPT_BITMAP_MATCH_COUNT match, Silk4J stops taking snapshots and raises the exception E_BITMAP_NOT_STABLE.</p> <p>By default, this is 5.</p>
BitmapPixelTolerance	OPT_BITMAP_PIXEL_TOLERANCE	INTEGER	<p>Specifies the number of pixels of difference below which two bitmaps are considered to match. If the number of pixels that are different is smaller than the number specified with this option, the bitmaps are considered identical. The maximum tolerance is 32767 pixels.</p> <p>By default, this is 0.</p>
ButtonsToCloseWindows	OPT_CLOSE_WINDOW_BUTTONS	LIST OF STRING	Specifies the buttons used to close windows with the <code>CloseSynchronous</code> method.
ButtonsToConfirmDialogs	OPT_CLOSE_CONFIRM_BUTTONS	LIST OF STRING	Specifies the buttons used to close confirmation dialog boxes that appear when closing windows with the <code>CloseSynchronous</code> method.
CloseUnresponsiveApplications	OPT_KILL_HANGING_APPS	BOOLEAN	Specifies whether unresponsive applications are closed. An application is unresponsive if communication between the Agent and the application fails, e.g. times out. Set this option to TRUE when testing applications that cannot run multiple instances. By default, this is FALSE.
CloseWindowTimeout	OPT_CLOSE_WINDOW_TIMEOUT	NUMBER	Specifies the number of milliseconds to wait before the next close strategy is tried. The Agent executes four close attempts before failing, so the total time before a close fails is four times the value you specify.
Compatibility	OPT_COMPATIBILITY	STRING	<p>Enables you to use the Silk4J behavior of the specified Silk4J version for specific features, when the behavior of these features has changed in the latest version.</p> <p>Example strings:</p> <ul style="list-style-type: none"> 12 11.1 13.0.1

.NET Option Name	Agent Option Name	Constant Type	Description
			By default, this option is not set.
EnsureObjectIsActive	OPT_ENSURE_ACTIVE_OBJDEF	BOOLEAN	Ensures that the target object is active. By default, this is FALSE.
EnableAccessibility	OPT_ENABLE_ACCESSIBILITY	BOOLEAN	<p>TRUE to enable Accessibility when you are testing a Win32 application and Silk4J cannot recognize objects. Accessibility is designed to enhance object recognition at the class level.</p> <p>FALSE to disable Accessibility.</p> <p>By default, this is FALSE.</p> <p> Note: For Mozilla Firefox and Google Chrome, Accessibility is always activated and cannot be deactivated.</p>
EnableMobileWebview FallbackSupport	OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT	BOOLEAN	<p>Enables mobile native fallback support for hybrid mobile applications that are not testable with the default browser support.</p> <p>By default, this is FALSE.</p>
EnableUiAutomationSupport	OPT_ENABLE_UI_AUTOMATION_SUPPORT	NUMBER	<p>TRUE to enable Microsoft UI Automation support instead of the normal Win32 control recognition. This option might be useful when you are testing a Win32 application and Silk4J cannot recognize objects.</p> <p>AUTODETECT to automatically enable Microsoft UI Automation support for JavaFX.</p> <p>By default, this is FALSE.</p>
HangAppTimeOut	OPT_HANG_APP_TIME_OUT	NUMBER	<p>Specifies the Unresponsive application timeout, which is the timeout for pending playback actions.</p> <p>The default value is 5000 milliseconds.</p>
HighlightObjectDuring Playback	OPT_REPLAY_HIGHLIGHT	BOOLEAN	<p>Specifies whether the current object is highlighted during playback.</p> <p>By default, this is FALSE, which means that objects are not highlighted by default.</p>
KeyboardEventDelay	OPT_KEYBOARD_INPUT_DELAY	NUMBER	<p>Sets the delay in milliseconds between playback of keyboard strokes.</p> <p>Be aware that the optimal number you select can vary, depending on the application that you are testing. For example, if you are testing a Web</p>

.NET Option Name	Agent Option Name	Constant Type	Description
			application, a setting of 1 millisecond radically slows down the browser. However, setting this to 0 (zero) may cause basic application testing to fail.
KeysToCloseDialogs	OPT_CLOSE_DIALOG_KEYS	LIST OF STRING	Specifies the keystroke sequence to close dialog boxes that open after trying to close a window with the <code>CloseSynchron</code> method. Examples include: <ESC>, <Alt+F4>.
LocatorAttributesCase Sensitive	OPT_LOCATOR_ATTRIBUTES_CASE_SENSITIVE	BOOLEAN	Set to <code>Yes</code> to add case-sensitivity to locator attribute names, or to <code>No</code> to match the locator names case insensitive. The names of locator attributes for mobile Web applications are always case insensitive, and this option is ignored when recording or replaying mobile Web applications.
MenuItemsToCloseWindows	OPT_CLOSE_WINDOW_MENUS	LIST OF STRING	Specifies the menu items used to close windows with the <code>CloseSynchron</code> method. Examples include: "File/Exit*", "File/Quit*".
MouseEventDelay	OPT_MOUSE_INPUT_DELAY	NUMBER	Specifies the delay in milliseconds used before each mouse event.
ObjectResolveRetryInterval	OPT_WAIT_RESOLVE_OBJDEF_RETRY	NUMBER	Specifies the time in milliseconds to wait before re-trying to find an object that could not be immediately resolved during playback. When the <code>ObjectResolveTimeout</code> runs out, no further retries are attempted.
ObjectResolveTimeout	OPT_WAIT_RESOLVE_OBJDEF	NUMBER	Specifies the time in milliseconds to wait for an object to be resolved during playback. As soon as the object is resolved, Silk4J can recognize it.
PlaybackMode	OPT_REPLAY_MODE	NUMBER	Defines how controls are replayed. Use low level to replay each control using the mouse and keyboard. Use high level to use the API to replay each control. All controls have a default playback mode assigned. When the default replay mode is selected, each control uses its default playback mode. The default mode delivers the most reliable results. Selecting low or high level playback overrides the playback mode of all controls with the playback mode selected.

.NET Option Name	Agent Option Name	Constant Type	Description
PostReplayDelay	OPT_POST_REPLAY_DELAY	NUMBER	<p>Possible values include 0, 1 and 2. 0 is default, 1 is high level, 2 is low level. By default, this is 0.</p> <p>Specifies the time in milliseconds to wait after invoking a function or setting a property.</p>
RemoveFocusOnCaptureText	OPT_REMOVE_FOCUS_ON_CAPTURE_TEXT	BOOLEAN	<p>Set to Yes to take the focus off the application under test during a text capture. By default, this is set to No, leaving the focus on the application under test. A text capture is performed during recording and replay by the following methods:</p> <ul style="list-style-type: none"> • <code>TextClick</code> • <code>TextCapture</code> • <code>TextExists</code> • <code>TextRect</code>
SyncTimeout	OPT_SYNC_TIMEOUT	NUMBER	<p>Specifies the maximum time in milliseconds for an object to be ready.</p> <p> Note: When you upgrade from a Silk Test version prior to Silk Test 13.0, and you had set the <code>OPT_XBROWSER_SYNC_TIMEOUT</code> option, the Options dialog box will display the default value of the <code>OPT_SYNC_TIMEOUT</code>, although your timeout is still set to the value you have defined.</p>
TransparentClasses	OPT_TRANSPARENT_CLASSES	LIST OF STRING	<p>To simplify the object hierarchy and to shorten the length of the lines of code in your test scripts and functions, you can suppress the controls for certain unnecessary classes in the following technologies:</p> <ul style="list-style-type: none"> • Win32. • Java AWT/Swing. • Java SWT/Eclipse. • Windows Presentation Foundation (WPF). <p>Specify the names of any classes that you want to ignore during recording and playback.</p>
	OPT_WPF_CHECK_DISPATCHER_FOR_IDLE	BOOLEAN	<p>For some WPF applications the Silk Test synchronization might not work due to how certain controls are implemented, resulting in Silk4J not</p>

.NET Option Name	Agent Option Name	Constant Type	Description
			recognising when the WPF application is idle. Setting this option to FALSE disables the WPF synchronization and prevents Silk4J from checking the WPF dispatcher, which is the thread that controls the WPF application. Set this option to FALSE to solve synchronization issues with certain WPF applications. By default, this is TRUE.
WPFCustomClasses	OPT_WPF_CUSTOM_CLASSES	LIST OF STRING	Specify the names of any WPF classes that you want to expose during recording and playback. For example, if a custom class called <i>MyGrid</i> derives from the WPF <i>Grid</i> class, the objects of the <i>MyGrid</i> custom class are not available for recording and playback. <i>Grid</i> objects are not available for recording and playback because the <i>Grid</i> class is not relevant for functional testing since it exists only for layout purposes. As a result, <i>Grid</i> 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 <i>MyGrid</i> , 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.
WPFPrefillItems	OPT_WPF_PREFILL_ITEMS	BOOLEAN	Defines whether items in a <i>WPFItemsControl</i> , like <i>WPFComboBox</i> or <i>WPFListBox</i> , are pre-filled during recording and playback. WPF itself lazily loads items for certain controls, so these items are not available for Silk4J if they are not scrolled into view. Turn pre-filling on, which is the default setting, to additionally access items that are not accessible without scrolling them into view. However, some applications have problems when the items are pre-filled by Silk4J in the background, and these applications can therefore crash. In this case turn pre-filling off.
XbrowserEnableIframe Support	OPT_XBROWSER_ENABLE_IFRAME_SUPPORT	BOOLEAN	Specifies whether to enable iframe and frame support for browsers. If you are

.NET Option Name	Agent Option Name	Constant Type	Description
			not interested in the content of the iframes in a web application, disabling the iframe support might improve replay performance. For example, disabling the iframe support might significantly improve replay performance for web pages with many ads and when testing in a mobile browser. This option is ignored by Internet Explorer. This option is enabled by default.
XBrowserExcludeIframes	OPT_XBROWSER_EXCLUDE_IFRAMES	STRING	Every entry in the list defines an attribute name and the corresponding value. All iframes and frames that do not match at least one of the entries are considered during testing. Wildcards are allowed, for example the entry "src:*advertising*" would exclude <IFRAME src=http://my.domain/advertising-banner.html>. This option is ignored by Internet Explorer. If the list is empty, all iframes and frames are considered during testing. Separate multiple entries with a comma.
XBrowserFindHiddenInputFields	OPT_XBROWSER_FIND_HIDDEN_INPUT_FIELDS	BOOLEAN	Specifies whether to display hidden input fields, which are HTML fields for which the tag includes type="hidden". The default value is TRUE.
XBrowserIncludeIframes	OPT_XBROWSER_INCLUDE_IFRAMES	STRING	Every entry in the list defines an attribute name and the corresponding value. All iframes and frames that do not match at least one of the entries are excluded. Wildcards are allowed, for example the entry "name:*form" would include <IFRAME name="user-form" src=...>. This option is ignored by Internet Explorer. If the list is empty, all iframes and frames are considered during testing. Separate multiple entries with a comma.
XBrowserSynchronizationMode	OPT_XBROWSER_SYNC_MODE	STRING	Configures the supported synchronization mode for HTML or 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. Using the AJAX mode eliminates the need to

.NET Option Name	Agent Option Name	Constant Type	Description
XBrowserSynchronizationTimeout	OPT_XBROWSER_SYNC_TIMEOUT	NUMBER	<p>manually script synchronization functions. By default, this value is set to AJAX.</p> <p>Specifies the maximum time in milliseconds for an object to be ready.</p> <p> Note: Deprecated. Use the option OPT_SYNC_TIMEOUT instead.</p>
XBrowserSynchronizationURLExcludes	OPT_XBROWSER_SYNC_EXCLUDE_URLS	STRING	<p>Specifies the URL for the service or Web page that you want to exclude during page synchronization. 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.</p> <p>Type the entire URL or a fragment of the URL, such as <i>http://test.com/timeService</i> or <i>timeService</i>.</p> <p>Separate entries by comma, for example:</p> <pre>desktop.setOption(Commonoptions.XBrowserSynchronizationURLExcludes, { "fpdownload.macromedia.com", "fpdownload.adobe.com", "download.microsoft.com" });</pre>

Configuring the Connections Between the Silk4J Components

To enable connecting to a remote machine through a firewall or to enable connecting to a remote machine securely by using HTTPS, you can configure the ports through which Silk4J communicates with the information service and the Open Agent.

When the Open Agent starts, a random available port is assigned to Silk4J and to the application that you are testing. The port numbers are registered on the *Silk Test information service* (information service).

The information service provides the following information to Silk4J:

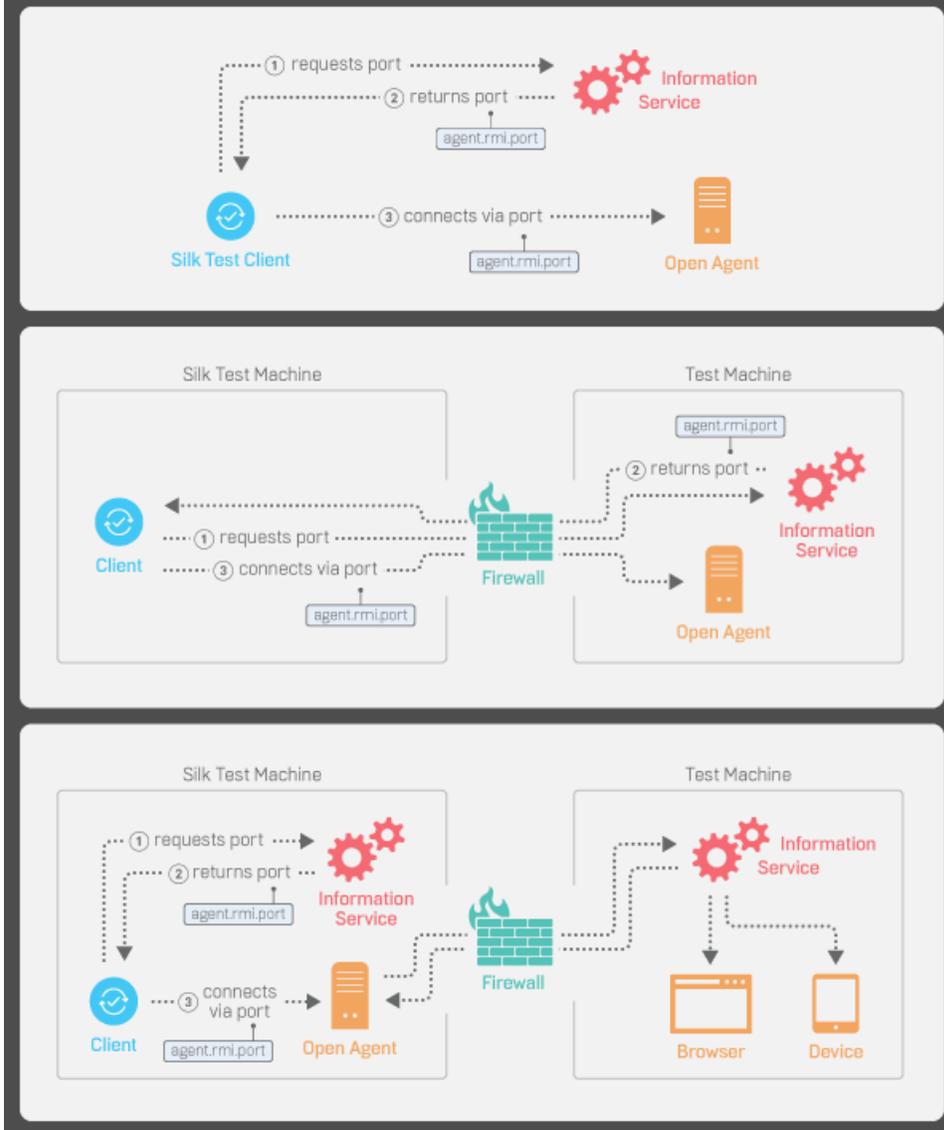
- The number of the port Silk4J can use to connect to the Open Agent. Communication runs directly between Silk4J and the agent. You might need to configure this port for remote agent scenarios, for example to avoid firewall conflicts.
- The browsers that are available on the machine on which the information service is installed.
- The mobile devices that are connected to the machine on which the information service is installed.
- The emulators that are available on the machine on which the information service is installed.
- The mobile browsers that are available on the previously mentioned mobile devices and emulators.

By default, the Open Agent communicates with the information service on HTTPS port 48561. 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 want to test on a remote machine and there is a port number conflict or an issue with a firewall between the machine on which Silk4J is installed and the test machine, you can configure the port number for the communication between Silk4J and the Open Agent on the remote machine or the port number for the communication between Silk4J and the information service on the remote machine. If you have multiple remote machines on which you want to test, you can use different port numbers for each remote machine or you can use the same available port numbers for all remote machines.

The following image shows the communication between Silk4J, the information service and the Open Agent.

Silk Test Client/Open Agent Connection



Configuring the Port 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 Open Agent. Then, the information service forwards communication to the port that the Open Agent uses. However, you can configure the information service port settings to avoid problems with a firewall by forcing communication on a specific port.

By default, the port that is used to connect Silk4J with the information service over a secure HTTPS connection is port 48561. 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.

- In a Microsoft Windows system, this file is located in `C:\ProgramData\Silk\Silk Test\conf`, where “`C:\ProgramData`” is equivalent to the content of the `ALLUSERSPROFILE` environment variable, which is set by default on Windows systems.
- On macOS, this file is located in `/Users/<user>/.silk/silktest/conf`.

This file contains commented text and sample alternate port settings.

2. Specify whether Silk4J should communicate with the information service over a secure connection through HTTPS.

- To use a secure connection through HTTPS, set `infoservice.https.enabled` to `true`. This is the default setting.
- To disable using a secure connection through HTTPS, set `infoservice.https.enabled` to `false`.

3. *Optional:* If you have specified that you want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.https.port`.

The default HTTPS port is 48561. Port numbers can be any number from 1 to 65535.

4. *Optional:* If you have specified that you do not want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.port`.

The default port is 22901.

5. Save the file as `infoservice.properties`.

6. Restart the Open Agent, the Silk Test client, and the application that you want to test.

Configuring the Port 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 Open Agent. Then, the information service forwards communication to the port that the Open Agent uses. However, you can configure the information service port settings to avoid problems with a firewall by forcing communication on a specific port.

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

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

By default, this file is located at: `%APPDATA%\Silk\SilkTest\conf`, which is typically `C:\Users\<user name>\AppData\Silk\SilkTest\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.
- `agent.heartbeat.port` – Required to test with an Open Agent that is installed on a remote machine.



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

3. Save the file as `agent.properties`.
4. Restart the Open Agent, the Silk Test client, and the application that you want to test.

Editing the Properties of the Silk Test Information Service

Use the `infoservice.properties` file to specify the port for the Silk Test Information Service, whether to use a secure connection through HTTPS, or the capabilities that are applied each time Silk Test executes a test on the machine on which the Silk Test Information Service is running.

1. Navigate to the directory in which the `infoservice.properties.sample` file is located.
 - On a Windows machine, navigate to `%PROGRAMDATA%\Silk\SilkTest\conf`, for example `C:\ProgramData\Silk\SilkTest\conf`.
 - On macOS, navigate to `~/ .silk/silktest/conf/`.
2. Rename the file `infoservice.properties.sample` to `infoservice.properties`.
3. Specify whether Silk4J should communicate with the information service over a secure connection through HTTPS.
 - To use a secure connection through HTTPS, set `infoservice.https.enabled` to `true`. This is the default setting.
 - To disable using a secure connection through HTTPS, set `infoservice.https.enabled` to `false`.
4. *Optional:* If you have specified that you want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.https.port`.
The default HTTPS port is 48561. Port numbers can be any number from 1 to 65535.
5. *Optional:* To redirect all HTTP requests to the HTTPS port, if you have specified that you want to use a secure connection through HTTPS, set `infoservice.http-to-https.enabled` to `true`.
The default value is `false`.
6. *Optional:* If you have specified that you do not want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.port`.
The default port is 22901.
7. *Optional:* To replace the certificates that are used by Silk Test for the HTTPS connection with your own certificates, see [Replacing the Certificates that are Used for the HTTPS Connection to the Information Service](#).

8. To specify capabilities, add the following line to the `infoservice.properties` file:

```
customCapabilities=<custom_capability_1>;<custom_capability_2>;...
```

Example: Running an iOS Simulator in a Specified Language

To always run a specific iOS Simulator on a Mac in the same language, for example Japanese, specify the custom capabilities *language* and *locale*. To do so, add the following line to the `infoservice.properties` file:

```
customCapabilities=language=ja;locale=ja_JP
```

Replacing the Certificates that are Used for the HTTPS Connection to the Information Service

When using a secure connection through HTTPS between Silk4J and the information service, the following self-signed certificate files are used:

- The `keystore` certificate file is used for the information service HTTPS server.
- The following certificate files are used for the machine on which the Silk4J client is running:
 - `cacerts`
 - `cacerts.p12`
 - `cacerts.pem`

You can use OpenSSL and the Java `keytool` executable to replace these files with your own certificate files.

1. Ensure that OpenSSL and a JDK are installed on your machine.
2. Start the Java `keytool` executable from the `bin` folder of your JDK installation folder.
3. Create a private and public key pair in your private `keystore` file on the information service HTTPS server:

```
keytool -genkey -alias jetty -keyalg RSA -keypass Borland -storepass Borland -keystore keystore -validity 1095
```

4. When prompted to type a first and last name, type `*` as a wildcard for the host.
5. Export the information from your private `keystore` to a temporary certificate file named `server.cer`:

```
keytool -export -alias jetty -storepass Borland -file server.cer -keystore keystore
```

This temporary certificate file is required to generate the certificate files for the machine on which the Silk4J client is running.

6. Create a certificate file named `cacerts` from the `server.cer` file.

```
keytool -import -v -trustcacerts -alias jetty -file server.cer -keystore cacerts -keypass Borland -storepass Borland
```

7. Import the information from the `cacerts` file into the temporary certificate file `cacerts.p12`.

```
keytool -importkeystore -srckeystore cacerts -destkeystore cacerts.p12 -srcstoretype JKS -deststoretype PKCS12 -srcstorepass Borland -deststorepass Borland
```

8. Create the public keystore file `cacerts.pem` in the PKCS12 keystore format from the temporary certificate file `cacerts.p12`

```
openssl pkcs12 -in cacerts.p12 -out cacerts.pem -cacerts -nokeys
```

9. Deploy the files `keystore`, `cacerts`, `cacerts.p12`, and `cacerts.pem` to the configuration folder:

- On a Windows machine, deploy the files to `%PROGRAMDATA%\Silk\SilkTest\conf`, for example `C:\ProgramData\Silk\SilkTest\conf`.

- On macOS, deploy the files to `~/ .silk/silktest/conf/`.

Remote Testing with the Open Agent

You can install Silk4J on a remote machine and test an application on this remote location from the Silk4J that is installed on your local machine.



Note: If you want to test mobile applications on a mobile device that is connected to a remote machine or on an Emulator or Simulator on a remote machine, or web applications on Apple Safari or on a remote Microsoft Edge, you have to use a remote Silk Test information service instead of a remote Open Agent.

Testing with a Remote Open Agent

To replay tests against an application on a remote machine with Silk4J, perform the following actions:

1. Create a test against the application on the local machine.
2. Install the Open Agent on the remote machine.
For additional information, refer to the [Silk Test Installation Guide](#).
3. Start the Open Agent on the remote machine.
4. In the test script, specify the remote machine as a new desktop.
For example, add the following line to the script:

```
private Desktop desktop = new Desktop("<IP address or network name of  
remote machine>");
```

Configuring the Open Agent to Run Remotely in a NAT Environment

To remotely run the Open Agent in a network address translation (NAT) environment, for example a virtual machine (VM), configure the agent to include a VM argument.

1. Navigate to the `agent.properties.sample` file and open it.
By default, this file is located at: `%APPDATA%\Silk\SilkTest\conf`, which is typically `C:\Documents and Settings\<user name>\Application Data\Silk\SilkTest\conf`.
2. Add the following property:

```
java.rmi.server.hostname=<external IP of VM>
```
3. Save the file as `agent.properties`.

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 an application, 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



Note: Do not add more than one browser application configuration when testing a web application with a defined base state.

You can edit the base state in the following ways:

- Through the user interface, for example if you want to specify how to start the AUT during both recording and replay.
- In a script, for example if you only want to start the AUT in the specified way when replaying the tests in the script.

Modifying the Base State from the User Interface

You can edit the base state from the user interface to specify how Silk4J starts an application under test (AUT) during recording and replay. You can specify the executable location of the AUT, the working directory, the URL and the connection string for a web application, and so on. For example, if you want to execute tests on a production web site, which have already been executed on a staging web site, you can simply change the URL in the base state and the tests are executed against the new web site.



Note: To specify how Silk4J starts an application under test (AUT) only for specific tests during replay, edit the base state in the script that contains the tests. For additional information, see [Modifying the Base State in a Script](#).

To edit the base state through the user interface:

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** to the right of the application configuration that you want to change.
3. To test a web application or a mobile web application, if you have not set an application configuration for the current project, select one of the installed browsers or mobile browsers from the list.
You can also click **Change** to open the **Select Application** dialog box and then select the browser that you want to use.
4. To specify an executable, type the full path to the executable into the **Executable** field.



Note: If you are testing a web application, and you want to specify an executable for the browser, select the custom browser type.

For example, to start Mozilla Firefox, type `C:\Program Files (x86)\Mozilla Firefox\firefox.exe`.

5. If you are testing a desktop application and you want to use an executable pattern, type the executable name and file path of the desktop application that you want to test into the **Executable Pattern** text box. For example, you might type `*\calc.exe` to specify the Calculator.

6. If you are testing a desktop application and you want to use a command line pattern in combination with the executable file, type the command line pattern into the **Command Line Pattern** text box.

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.

7. To specify command line arguments, type the arguments into the **Command Line Arguments** field.



Note: If you are testing a web application, and you want to start a browser with command line arguments, select the custom browser type.

For example, to start Mozilla Firefox with the profile `myProfile`, type `-p myProfile`.

8. If you are testing an application which depends on a supplemental directory, specify the path to the directory in the **Working directory** field.

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.

9. If you are testing a desktop application, specify the main window of the application in the **Locator** field. For example, the locator might look like `/Shell[@caption='Swt Test Application']`.

10. If you are testing a web application, type the address of the web application into the **Navigate to URL** text box.

11. *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.

For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.

12. *Optional:* Select an **Orientation** for the browser window.

13. *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.

14. If you are testing a web application or a mobile native application on a remote location, for example on a mobile device that is connected to a Mac, and you want to edit the remote location, click **Change** to open the **Select Application** dialog box and then click **Edit Remote Locations**.

15. If you are testing a mobile application or a web application on Apple Safari, type the connection string into the **Connection String** text box.

For additional information, see [Connection String](#).

16. To edit the capabilities for a WebDriver-based browser, you can use the **Connection String** text box. For example, to start Google Chrome with a maximized browser window, type the following into the **Connection String** text box:

```
chromeOptions={"args":["--start-maximized"]}
```

For additional information, see [Setting Options and Capabilities for Web-Driver Based Browsers](#).

17. If you are testing a mobile native application, specify the application:

- If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
- If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.

- If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
- If you want to use an app that is available in Mobile Center, specify the **App identifier**.

18. Click **OK**.

19. If the application under test usually takes a long time to start, increase the application ready timeout in the replay options.

Executing the base state starts the application if it is not already running. If the application is already running, Silk4J does not start another instance of the application.

If your test includes multiple application configurations and you are modifying an application or Web page other than the object associated with the base state, you can turn off the base state. This indicates that the base state will not be used for recording or replaying the modifications. Therefore, you must record the steps to launch the application or Web page within your test. For instance, if you want to test a Web page, start Internet Explorer within your test.

 **Note:** Do not add more than one browser application configuration when testing a web application with a defined base state.

Modifying the Base State in a Script

You can edit the base state in a script to specify how Silk4J starts an application under test (AUT) during replay. You can specify the executable location of the AUT, the working directory, the URL and the connection string for a web application, and so on. For example, if you want to execute tests on a production web site, which have already been executed on a staging web site, you can simply change the URL in the base state and the tests are executed against the new web site.

 **Note:** To specify how Silk4J starts an application under test (AUT) during recording and replay, edit the base state from the user interface. For additional information, see *Modifying the Base State from the User Interface*.

To edit the base state in a script:

1. Open the script.
2. Change the `baseState` method.

You can add your code between creating the base state and executing it:

```
// Go to web page 'demo.borland.com/InsuranceWebExtJS'
BrowserBaseState baseState = new BrowserBaseState();
// <-- Insert your changes here!
baseState.execute();
```

3. Use the following code to specify the executable name and file path of the application that you want to test:

```
baseState.setExecutable(executable);
```

For example, to specify the Calculator, type the following:

```
baseState.setExecutable("C:\\Windows\\SysWOW64\\calc.exe");
```

To specify Mozilla Firefox, type the following:

```
baseState.setExecutable("C:\\Program Files (x86)\\Mozilla Firefox\\
\\firefox.exe");
```

4. Use the following code to specify command line arguments:

```
baseState.setCommandLineArguments(commandLineArguments);
```

For example, to start Mozilla Firefox with the profile *myProfile*, type the following:

```
baseState.setCommandLineArguments("-p myProfile");
```

5. You can use the following code to specify a different working directory:

```
baseState.setWorkingDirectory(workingDirectory);
```

6. If you want to use an executable pattern, use the following code:

```
baseState.setExecutablePattern(executablePattern);
```

For example, if you want to specify an executable pattern for the Calculator, type:

```
baseState.setExecutablePattern("*\\calc.exe");
```

7. If you want to use a command line pattern in combination with the executable file, use the following code:

```
baseState.setCommandLinePattern(commandLinePattern);
```

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:

```
baseState.setCommandLinePattern("*com.example.MyMainClass");
```

8. If you are testing a web application or a mobile web application, and you have not set an application configuration for the current project, specify one of the installed browsers or mobile browsers. For example, to specify Google Chrome, type the following:

```
baseState.setBrowserType(BrowserType.GoogleChrome);
```

9. If you are testing a web application or a mobile web application, and you have not set an application configuration for the current project, specify the address of the web application that you want to test:

```
baseState.setUrl(url);
```

For example, type the following:

```
baseState.setUrl("demo.borland.com/InsuranceWebExtJS/");
```

10. If you want to test a web application on a desktop browser, you can specify the height and width of the browser window:

```
baseState.setViewportHeight(viewportHeight);
baseState.setViewportWidth(viewportWidth);
```

11. If you want to test a web application or a mobile native application on a remote location, specify the connection string:

```
new MobileBaseState(connectionString);
```

For information on the connection string, see [Connection String for a Remote Desktop Browser](#) or [Connection String for a Mobile Device](#).

12. To edit the capabilities for Mozilla Firefox or Google Chrome, you can also use the connection string. For example, to set the download folder for Mozilla Firefox, type the following:

```
baseState.setConnectionString(
    "moz:firefoxOptions="
    + "{
    + \"prefs\": {
    + \"browser.download.dir\": \"C:\\\\Download\\\\\\\\\"
    + }
    + };" );
```

For additional information, see [Setting Options and Capabilities for Web-Driver Based Browsers](#).

Running the Base State

Before starting to record a test against an application, you can execute the base state to bring all applications, against which you want to record, to the appropriate state for recording.

To run the base state:

Click **Silk4J > Run Base State**.

Depending on the type of the application, Silk4J performs the following actions:

- Executes the application configurations of all applications, for which an application configuration is defined in the current project.
- For desktop applications, Silk4J opens the application from the specified installation directory. If the specified path to the directory includes the `%ProgramFiles%` environment variable, and Silk4J cannot find the application in the `Program Files` directory, Silk4J additionally searches in the `Program Files (x86)` directory. If the specified path to the directory includes the `%ProgramFiles(x86)%` environment variable, Silk4J first searches in the `Program Files (x86)` directory, and if the application is not found, Silk4J additionally searches the `Program Files` directory.
- For web applications, Silk4J opens the web application in the specified browser and to the specified URL.
- For mobile web applications, Silk4J opens the specified browser on the specified mobile device or Emulator to the specified URL.
- For mobile native applications, Silk4J opens the specified app on the specified mobile device or Emulator. If the specified app is not installed on the specified mobile device or Emulator, Silk4J installs and then opens the app.

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.

- For a Windows application, an application configuration includes the following:
 - Executable pattern
All processes that match this pattern are enabled for testing. For example, the executable pattern for Internet Explorer is `*\IEXPLORE.EXE`. All processes whose executable is named `IEXPLORE.EXE` and that are located in any arbitrary directory are enabled.
 - Command line pattern
The command line pattern is an additional pattern that is used to constrain the process that is enabled for testing by matching parts of the command line arguments (the part after the executable name). An application configuration that contains a command line pattern enables only processes for testing that match both the executable pattern and the command line pattern. If no command-line pattern is defined, all processes with the specified executable pattern are enabled. Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, `*\javaw.exe` is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you want is enabled for testing. For example, if the command line of the application ends with `com.example.MyMainClass` you might want to use `*com.example.MyMainClass` as the command line pattern.
- For a web application in a desktop browser on the local machine, an application configuration includes only the browser type.
 -  **Note:** To start a browser with command line arguments or to specify a working directory or a specific executable for the browser, select the custom browser type. For additional information, see *Modifying the Base State*.
- For a web application in Apple Safari or in Microsoft Edge on a remote machine, an application configuration includes the following:
 - Browser type.
 - Connection string.
- For a web application in a mobile browser, an application configuration includes the following:
 - Browser type.
 - Connection string.
- For a native mobile application, an application configuration includes the following:
 - Connection string.
 - Simple application name. If multiple applications on the mobile device have the same name, the fully qualified name of the application is used.

 **Note:** Do not add more than one browser application configuration when testing a web application with a defined base state.

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 application configuration**.

 **Note:** Do not add more than one browser application configuration when testing a web application with a defined base state.

The **Select Application** dialog box opens. Select the tab and then the application that you want to test and click **OK**.

3. To remove an application configuration, click **Remove** next to the appropriate application configuration.
4. To edit an application configuration, click **Edit** next to the appropriate application configuration.
5. To specify an executable, type the full path to the executable into the **Executable** field.

 **Note:** If you are testing a web application, and you want to specify an executable for the browser, select the custom browser type.

For example, to start Mozilla Firefox, type `C:\Program Files (x86)\Mozilla Firefox\firefox.exe`.

6. If you are testing a desktop application, type the name of the executable of the application that you want to test into the **Executable Pattern** field.

The pattern is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

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.

7. If you are testing a desktop application and you want to use a command line pattern in combination with the executable file, type the command line pattern into the **Command Line Pattern** text box.
8. To test a web application or a mobile web application, if you have not set an application configuration for the current project, select one of the installed browsers or mobile browsers from the list.

You can also click **Change** to open the **Select Application** dialog box and then select the browser that you want to use.

9. If you are testing a web application, type the address of the web application into the **Navigate to URL** text box.

10. If you are testing a web application against Mozilla Firefox or Google Chrome, and a proxy is used for the connection with the web, specify the proxy in the **Proxy** field.

Setting the proxy is useful when you are using service virtualization. You can use the following formats.

Format	Description	Examples
<code><machine_name>:<port></code>	Specifies a proxy for all traffic.	<code>localhost:9000</code>
<code>http=<machine_name>:<port></code> or <code>http://<machine_name>:<port></code>	Specifies a proxy for HTTP URLs.	<code>http=localhost:9000</code> or <code>http://localhost:9000</code>

Format	Description	Examples
https=<machine_name>:<port> or https://<machine_name>:<port>	Specifies a proxy for HTTPS URLs.	https=localhost:9000 or https://localhost:9000
http=<machine_name>:port;https=<another_machine_name>:<another_port> or http://<machine_name>:port;https://<another_machine_name>:<another_port>	Specifies different proxy servers for HTTP URLs and HTTPS URLs.	http=localhost:8000;https=my_HTTPS_host:9000 or http://localhost:8000;https://my_HTTPS_host:9000

11. To detect any visual breakpoints for a web application, click **Edit Browser Sizes**.
For additional information, see *Detecting Visual Breakpoints*.
12. *Optional*: If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
13. *Optional*: Select an **Orientation** for the browser window.
14. *Optional*: Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.
15. If you are testing a mobile application or a web application on Apple Safari, type the connection string into the **Connection String** text box.
For additional information, see *Connection String for a Mobile Device*.
16. Click **OK**.

Select Application Dialog Box

Use the **Select Application** dialog box to select the application that you want to test, to associate an application with an object map, or to add an application configuration to a test. Application types are listed in tabs on the dialog box. Select the tab for the application type you want to use.

Windows Lists all Microsoft Windows applications that are running on the system. Select an item from the list and click **OK**.

Use the **Hide processes without caption** check box to filter out applications that have no caption.

Web Lists all available browsers, including mobile browsers on any connected mobile devices and Apple Safari on a Mac. Specify the web page to open in the **Enter URL to navigate** text box. If an instance of the selected browser is already running, you can click **Use URL from running browser** to record against the URL currently displayed in the running browser instance. If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list. When selecting a browser size, you can also select an **Orientation** for the selected browser.



Note: Do not add more than one browser application configuration when testing a web application with a defined base state.

Mobile Lists all available mobile devices and all running Android emulators, including devices connected to a remote location. Select this tab to test mobile native applications. You can select to test the mobile application (app) that is currently running on the selected mobile device, or you can browse for or manually specify the name or file of the app that you want to test.

Editing Remote Locations

You can use the **Remote Locations** dialog box to add any browsers and mobile devices on a remote location to the set of applications that you can test.

1. Click **Silk4J > Edit Remote Locations**. The **Remote Locations** dialog box appears.
2. To add an additional remote location, perform the following actions:
 - a) Click on the arrow to the right of **Add Location** to specify whether you want to add a remote location which is using the Silk Test Information Service, Silk Central, or the Mobile Center.
 **Note:** You can only configure one Silk Central as a remote location. If you have already configured the integration with Silk Central, Silk Central is listed in the remote locations list.
 - b) Click **Add Location**. The **Add Location** dialog box appears.
 - c) Type the URL of the remote location and the port through which Silk4J connects to the information service on the remote machine into the **Host** field.
The default port is 22901.
 - d) If you are trying to add a secure connection through HTTPS to Mobile Center, install the Mobile Center certificate.
For additional information, see [Installing the Certificate for an HTTPS Connection to Mobile Center](#).
 - e) If you are trying to add connection to Mobile Center, specify your **User name** and **Password** for Mobile Center.
For information on changing the Mobile Center password, see [Changing the Mobile Center Password](#).
 - f) *Optional:* Edit the name of the remote location in the **Name** field.
 - g) *Optional:* Click **Test** to verify that the remote connection works.
 - h) Click **OK**.
3. To edit an existing remote location, click **Edit**.
4. To remove a remote location, click **Remove**.
5. *Optional:* To reduce the amount of browsers and devices in the **Select Application** dialog, click **Do not show devices and browsers from this location**. The installed browsers and connected devices of the remote location will no longer be displayed in the **Select Application** dialog. By default, all installed browsers and connected devices of all remote locations are displayed in the **Select Application** dialog.
6. By default, remote locations are saved to %APPDATA%\Silk\SilkTest\conf\remoteLocations.xml. To change the file in which the remote locations are saved, for example to share the file with other team members, perform the following actions:
 - a) Click **Sharing Options**. The **Sharing Options** dialog box appears.
 - b) Click **Save to shared file**.
 - c) Click **Browse** to select the file.
You could also set the full path to the remote locations in a file named `remoteLocationsFileLocation.properties`, for example if you want to use a file that is located in a repository. For example, to use the remote locations that are specified in the file `remoteLocations.xml`, under `C:\mySources`, create a file with the name `remoteLocationsFileLocation.properties` in the folder `%APPDATA%\Silk\SilkTest\conf\` and type `fileLocation=C:\mySources\remoteLocations.xml` into the file.
7. To load a set of already configured remote locations from a shared file, for example to use remote locations that have been specified by another team member, perform the following actions:
 - a) Click **Sharing Options**. The **Sharing Options** dialog box appears.
 - b) Click **Load from shared file**.
 - c) Click **Browse** to select the file.

8. Click **OK**.

When you have added a remote location, the browsers that are installed on the remote location, including Apple Safari on a Mac, are available in the **Web** tab of the **Select Application** dialog box, and the mobile devices that are connected to the remote location are available in the **Mobile** tab of the **Select Application** dialog box.

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	<ul style="list-style-type: none">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.
<ul style="list-style-type: none">The Select Browser dialog box does not display when running a test against a Web application.Multiple browser instances are started when running a test against a Web application.When running a test against a Web application with a browser instance open, Silk4J might stop working.	A base state and multiple browser application configurations are defined for the test case.	Remove all browser application configurations except one from the test case.
Playback error when running a test	No application configuration is defined for the test. The following exception might be displayed: <code>No application configuration present.</code>	<ul style="list-style-type: none">When running a keyword-driven test, ensure that a keyword which executes the base state is included in the test.Ensure that an application configuration is configured for the current project.

Troubleshooting Application Configurations

Why is my application not displayed in the **Select Application** dialog box

- Uncheck the **Hide processes without caption** check box. This check box is checked by default and prevents applications without a caption from being displayed in the dialog box.
- Run Silk4J with elevated privileges.

1. Close Silk4J.
 2. Stop the Open Agent.
 3. Run Silk4J as an administrator.
- Use the **Task Manager** to check if the application is running under a different user account.
 - Ensure that the application is not started with the `runas` command or a similar command.

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. If the test fails, because Silk4J cannot start the application, edit the application configuration.
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**.
 - b) In the **Executable Pattern** 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 Pattern** 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 `javaws.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.

Creating a Test that Tests Multiple Applications

You can test multiple applications with a single test script. To create such a test script, you need to add an application configuration for each application that you want to test to the project in which the script resides.

1. Record or manually script a test for the primary application that you want to test.
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. To add an additional application configuration, click **Add application configuration**.

 **Note:** Do not add more than one browser application configuration when testing a web application with a defined base state.

The **Select Application** dialog box opens. Select the tab and then the application that you want to test and click **OK**.

4. Click **OK**.

5. Record or manually script additional actions into the script using the new application configuration.



Note: Do not add more than one browser application configuration when testing a web application with a defined base state.

Setting Script Options

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

Setting TrueLog Options

You can enable TrueLog reports and HTML reports to capture bitmaps and to log information for test runs with Silk4J.

Logging bitmaps and controls might 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 the TrueLog Explorer, in the case of TrueLog reports, or in a browser, in the case of HTML reports. For additional information on the TrueLog Explorer, refer to the [Silk TrueLog Explorer Help for Silk Test](#).



Note: To reduce the size of TrueLog files with Silk Test 17.5 or later, the file format for TrueLog files has changed from `.xlg` to the compressed `.tlz` file format. Files with a `.xlg` suffix are automatically appended with a `.tlz` suffix. To parse result data from a `.tlz` file, you can unzip the `.tlz` file and parse the data from the included `.xlg` file.

To enable creating result data and to customize the information that Silk4J collects, perform the following steps:

1. Click **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **TrueLog** tab.
3. In the **Basic Settings** 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. Select the result format:
 - Select **TrueLog Report (.tlz)** to generate visual execution logs that can be viewed in TrueLog Explorer.
 - Select **HTML Report** to generate an HTML-based report that can be viewed in a browser.
 - Select **Both** to generate both a TrueLog report and an HTML-based report.
5. In the **TrueLog location** field, type the name of and optionally the path to the TrueLog file, or click **Browse** and select the file.

The 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 `.tlz` suffix. To ensure that TrueLog files are not overwritten, for example when you perform parallel testing, you can add placeholders to the TrueLog file name. These placeholders are replaced with the appropriate data at execution time. The HTML report is created in a sub-directory of the specified **TrueLog location**.



Note: The path is validated at execution time. Tests that are executed by Silk Central set this value to the Silk Central results directory to enable the screenshots to be shown in the result view.

6. Select the **Screenshot mode**.
Default is **On Error**.
7. *Optional:* Set the **Delay**.

This delay gives the operating system 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.

8. 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 **Silk4J > Edit Options**.
2. Click the **Recording** tab.
3. To set `Ctrl+Shift` as the shortcut key combination to use to pause recording, check the **OPT_ALTERNATE_RECORD_BREAK** check box.

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



Note: For SAP applications, you must set `Ctrl+Shift` as the shortcut key combination.

4. To record absolute values for scroll events, check the **OPT_RECORD_SCROLLBAR_ABSOLUT** check box.
5. To record mouse move actions for web applications, Win32 applications, and Windows Forms applications, check the **OPT_RECORD_MOUSEMOVES** check box. You cannot record mouse move actions for child technology domains of the xBrowser technology domain, for example Apache Flex and Swing.
6. If you record mouse move actions, in the **OPT_RECORD_MOUSEMOVE_DELAY** text box, specify how many milliseconds the mouse has to be motionless before a `MouseMove` is recorded.
By default this value is set to 200.
7. To record text clicks instead of `Click` actions on objects where `TextClick` actions usually are preferable to `Click` actions, check the **OPT_RECORD_TEXT_CLICK** check box.
8. To record image clicks instead of `Click` actions on objects where `ImageClick` actions usually are preferable to `Click` actions, check the **OPT_RECORD_IMAGE_CLICK** check box.
9. To define whether you want to record object map entries or XPath locators, select the appropriate recording mode from the **OPT_RECORD_OBJECTMAPS_MODE** list:
 - **Object map entries for new and existing objects.** This is the default mode.
 - **XPath locators for new and existing objects.**
 - **XPath locators for new objects only.** For objects that already exist in an object map, the object map entry is reused. Choosing this setting enables you to create object maps for the main controls of an AUT, and to persist these object maps while creating additional tests against the AUT.
10. To resize the application under test (AUT) when a recording session starts, check the **OPT_RESIZE_APPLICATION_BEFORE_RECORDING** check box.
This check box is checked by default, enabling the **Silk Recorder** to display next to the AUT. When this check box is unchecked, the AUT and the **Silk Recorder** might overlap.
11. 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 **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **Browser** tab.
3. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording. For example, if you do not want to record attributes named `height`, add the `height` attribute name to the grid.
Separate attribute names with a comma.
4. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording. For example, if you do not want to record attributes assigned the value of `x-auto`, add `x-auto` to the grid.
Separate attribute values with a comma.
5. To record native user input instead of DOM functions, check the **OPT_XBROWSER_RECORD_LOWLEVEL** check box.
For example, to record `Click` instead of `DomClick` and `TypeKeys` instead of `SetText`, check this check box.
If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.
6. To set the maximum length for locator attribute values, type the length into the field in the **Maximum attribute value length** section.
If the actual length exceeds that limit the value is truncated and a wild card (*) is appended. By default this value is set to 20 characters.
7. To automatically search for an unobstructed click spot on the specified target element, check the **OPT_XBROWSER_ENABLE_SMART_CLICK_POSITION** check box.
8. To force Mozilla Firefox to open external links in a new tab instead of a new window, check **OPT_FIREFOX_SINGLE_WINDOW_MODE**.



Note: This option only works with Mozilla Firefox 52 or later.

9. To disable iframe and frame support for browsers, uncheck **OPT_XBROWSER_ENABLE_IFRAME_SUPPORT**.
If you are not interested in the content of the iframes in a web application, disabling the iframe support might improve replay performance. For example, disabling the iframe support might significantly improve replay performance for web pages with many ads and when testing in a mobile browser. This option is ignored by Internet Explorer. This option is enabled by default.
10. In the **Whitelist for iframe support**, specify attributes of iframes and frames that should be considered during testing.
Every entry in the list defines an attribute name and the corresponding value. All iframes and frames that do not match at least one of the entries are excluded. Wildcards are allowed, for example the entry `"name:*form"` would include `<IFRAME name="user-form" src=...>`. This option is ignored by Internet Explorer. If the list is empty, all iframes and frames are considered during testing. Separate multiple entries with a comma.
11. In the **Blacklist for iframe support**, specify attributes of iframes and frames that should be excluded during testing.
Every entry in the list defines an attribute name and the corresponding value. All iframes and frames that do not match at least one of the entries are considered during testing. Wildcards are allowed, for example the entry `"src:*advertising*"` would exclude `<IFRAME src=http://my.domain/advertising-banner.html>`. This option is ignored by Internet Explorer. If the list is empty, all iframes and frames are considered during testing. Separate multiple entries with a comma.

12. Click **OK**.

Setting Custom Attributes

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

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

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

If multiple objects share the same attribute value, such as a caption, Silk4J tries to make the locator unique by combining multiple available attributes with the "and" operation and thus further narrowing down the list of matching objects to a single object. Should that fail, an index is appended. Meaning the locator looks for the *n*th 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 **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **Custom Attributes** tab.
3. From the **Select a tech domain** list box, select the technology domain for the application that you are testing.



Note: You cannot set custom attributes for Flex or Windows API-based client/server (Win32) applications.

4. Add the attributes that you want to use to the list.

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

Separate attribute names with a comma.



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



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



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

5. Click **OK**.

Setting Classes to Ignore

To simplify the object hierarchy and to shorten the length of the lines of code in your test scripts and functions, you can suppress the controls for certain unnecessary classes in the following technologies:

- Win32.
 - Java AWT/Swing.
 - Java SWT/Eclipse.
 - Windows Presentation Foundation (WPF).
1. Click **Silk4J > 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 **Silk4J > 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.



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

1. Click **Silk4J > 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.



Tip: Micro Focus recommends adding a substring of an URL to the exclude list, instead of the entire URL. For example, add `/syncsample` to the exclude list instead of `http://example.com/syncsample/timeService`. Excluding the entire URL might not work because the browser might return only a relative URL to Silk4J. For example, if the browser returns only `/syncsample/timeService` and you have added `http://example.com/syncsample/timeService` to the exclude list, Silk4J does not exclude the returned URL.

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 **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **Replay** tab. The **Replay Options** page displays.
3. If the application under test usually takes a long time to start, increase the time to wait for the application by increasing the value in the **OPT_APPREADY_TIMEOUT** text box.

4. 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 of the target technology. For example for Rumba controls, the Rumba RDE API is used to replay the controls.
 - **Low level** – Use this mode to replay each control using the mouse and keyboard.
5. To ensure that the object that you want to test is active, check the **OPT_ENSURE_ACTIVE_OBJDEF** check box.
6. To change the time to wait for an object to become enabled during playback, type the new time into the field in the **Object enabled timeout** section.
The time is specified in milliseconds. The default value is 1000.
7. To enable the **Playback Status** dialog box, check the **OPT_SHOW_PLAYBACK_STATUS_DIALOG** check box.
You can use the **Playback Status** dialog box to see the actions that are performed during the replay of a test on a remote location, for example when executing a test against a mobile application on a remote device.
8. To display a video or screenshots of the application under test in the **Playback Status** dialog box, check the **OPT_PLAYBACK_STATUS_DIALOG_SCREENSHOTS** check box.
9. To edit the prefix that specifies that an asset is located in the current project, edit the text for the **Asset namespace** option in the **OPT_ASSET_NAMESPACE** text box.
10. Click **OK**.

Setting UI Automation Options

Enable UI Automation support and specify which attributes and values to exclude from locators to allow Silk4J to better identify objects in applications that have implemented UI Automation provider interfaces, for example applications that are based on JavaFX or QT.



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

1. Click **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **UI Automation** tab.
3. Set **Enable Microsoft UI Automation Support** to *True* to enable Microsoft UI Automation support instead of the normal Win32 control recognition.
 -  **Note:** If you are testing against a Java FX application, you do not have to enable the UI Automation support, as Silk4J enables this out-of-the-box for Java FX applications.
 -  **Note:** The UI Automation support overrides the standard technology-domain-specific support. When you are finished interacting with the controls that require UI Automation support, disable the UI Automation support again to resume working with standard controls.
4. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording. For example, if you want to ignore the attribute name `automationid`, because multiple controls in your AUT have an attribute with this name, add the `automationid` attribute name to the list.
Separate attribute names with a comma.
5. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording. For example, the value `JavaFX*` is added to the list by default, because all JavaFX controls include an attribute value of the form `JavaFX<number>`.
Separate attribute names with a comma.
6. Click **OK**.

Setting Advanced Options

Set advanced options to enable fallback support, to specify whether locator attribute names should be case sensitive, and so on.

1. Click **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click the **Advanced** tab. The **Advanced Options** page displays.
3. To test an embedded Chrome application, specify the executable and the port as a value pair in the **Enable embedded Chrome support** field.

For example, `myApp.exe=9222`.

To specify multiple embedded Chrome applications, separate the value pairs with a comma.

4. Enable **OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT** to enable the mobile native fallback support for hybrid mobile applications that are not testable with the default browser support.
5. Enable **OPT_ENABLE_ACCESSIBILITY** to enable Microsoft Accessibility in addition to the normal Win32 control recognition.
6. Enable **OPT_REMOVE_FOCUS_ON_CAPTURE_TEXT** to remove the focus from the window before capturing a text.

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

- `TextClick`
- `TextCapture`
- `TextExists`
- `TextRect`

7. Enable **OPT_LOCATOR_ATTRIBUTES_CASE_SENSITIVE** to set locator attribute names to be case sensitive. The names of locator attributes for mobile web applications are always case insensitive, and this option is ignored when recording or replaying mobile web applications.
8. Set the default accuracy level for new image assets by selecting a value from 1 (low accuracy) to 10 (high accuracy) from the **OPT_IMAGE_ASSET_DEFAULT_ACCURACY** list box.
9. Set the default accuracy level for new image verification assets by selecting a value from 1 (low accuracy) to 10 (high accuracy) from the **OPT_IMAGE_VERIFICATION_DEFAULT_ACCURACY** list box.
10. 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**.

Converting Projects to and from Silk4J

A Silk4J project has the following additional characteristic as compared to a standard Java project:

- A dependency to the Silk4J library and the JUnit library.

Converting a Java Project to a Silk4J Project

If you have an existing Java project that you want to use with Silk4J, follow this procedure.

1. In the **Package Explorer**, right-click the Java project that you want to convert to a Silk4J project. The project context menu appears.
2. Choose **Silk4J Tools > Make Silk4J Project** .

The Silk4J library is added to the project. If the project does not contain a dependency to the JUnit library, this library is also added to the project.

Converting a Silk4J Project to a Java Project

1. In the **Package Explorer**, right-click the Silk4J project that you want to convert to a Java project. The project context menu appears.
2. Choose **Silk4J Tools > Remove Silk4J Capability** .

The Silk4J library is removed from the project.



Note: The dependency to the JUnit library remains in place since it is likely that this project will continue to use JUnit.

Testing Specific Environments

Silk4J supports testing several types of environments.

Active X/Visual Basic Applications

Silk4J provides support for testing ActiveX/Visual Basic applications.

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

Dynamically Invoking ActiveX/Visual Basic Methods

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

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

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

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

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

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



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



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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

Returned Values

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

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

Apache Flex Support

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

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

Supported Controls

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



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

Configuring Flex Applications to Run in Adobe Flash Player

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

- The developer who creates the Flex application must compile the application as an EXE file. When a user launches the application, it will open in Flash Player. Install Windows Flash Player from <http://www.adobe.com/support/flashplayer/downloads.html>.
 - The user must have Windows Flash Player Projector installed. When a user opens a Flex .SWF file, he can configure it to open in Flash Player. Windows Flash Projector is not installed when Flash Player is installed unless you install the Apache Flex developer suite. Install Windows Flash Projector from <http://www.adobe.com/support/flashplayer/downloads.html>.
1. For Microsoft Windows 7 and Microsoft Windows Server 2008 R2, configure Flash Player to run as administrator. Perform the following steps:
 - a) Right-click the Adobe Flash Player program shortcut or the `FlashPlayer.exe` file, then click **Properties**.
 - b) In the **Properties** dialog box, click the **Compatibility** tab.
 - c) Check the **Run this program as an administrator** check box and then click **OK**.
 2. Start the .SWF file in Flash Player from the command prompt (`cmd.exe`) by typing:

```
"<Application_Install_Directory>\ApplicationName.swf"
```

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

Launching the Component Explorer

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

In Internet Explorer, open <http://demo.borland.com/flex/SilkTest20.0/index.html>. The application launches in your default browser.

Testing Apache Flex Applications

Silk Test provides built-in support for testing Apache Flex applications. Silk Test also provides several sample Apache Flex applications. You can access the sample applications at <http://demo.borland.com/flex/SilkTest20.0/index.html>.

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

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

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

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

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



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

Testing Apache Flex Custom Controls

Silk4J supports testing Apache Flex custom controls. However, by default, Silk4J cannot record and playback the individual sub-controls 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 Apache 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 Apache Flex control.
- Seamless integration into Silk4J API
- Silk4J uses the specific class name in the locator, for example Silk4J records `//FlexSpinner`.

The disadvantages of advanced support include:

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

Dynamically Invoking Flex Methods

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

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

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



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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

Returned Values

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

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

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

Defining a Custom Control in the Test Application

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

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

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



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

The custom control offers a public "Value" 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,
downButtonClickListener);
        upButton.addEventListener(MouseEvent.CLICK,
upButtonClickListener);
        updateText();
    }

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

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

    private function upButtonClickListener(event : MouseEvent) : void {
        if(cValue <= upperBound - stepSize) {
            Value = Value + stepSize;
        }
        else {
            Value = lowerBound + Value + 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 Value() : int {
        return _value;
    }

    public function set Value(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(Value < lowerBound) {
        Value = lowerBound;
    }
}

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

public function set upperBound(v : int) : void {
    _upperBound = v;
    if(Value > upperBound) {
        Value = 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 "Value" 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("Value", 4);
```

Testing a Custom Control Using Automation Support

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

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

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

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

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

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

```
package customcontrols;  
  
import com.borland.silktest.jtf.Desktop;  
import com.borland.silktest.jtf.common.JtfObjectHandle;  
import com.borland.silktest.jtf.flex.FlexBox;  
  
/**  
 * Implementation of the FlexSpinner Custom Control.  
 */  
public class FlexSpinner extends FlexBox {  
  
    protected FlexSpinner(JtfObjectHandle handle, Desktop desktop) {  
        super(handle, desktop);  
    }  
}
```

```

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

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

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

public Integer getValue() {
    return (Integer) getProperty("Value");
}

public void setValue(Integer Value) {
    setProperty("Value", Value);
}

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']").setValue(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 "Value" 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.Value.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"?>
<TypeInfo>
    <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 Value property -->
  <Property Name="Value" accessType="both">
    <PropertyType Type="integer" />
  </Property>
  <Property Name="stepSize" accessType="read">
    <PropertyType Type="integer" />
  </Property>
</Properties>
</ClassInfo>
</TypeInfo>

```

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 Apache 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 Apache Flex, it reads the contents of this directory.

The XML file has the following basic structure:

```
<TypeInfo>
<ClassInfo>
<Implementation />
<Events>
<Event />
...
</Events>
<Properties>
<Property />
...
</Properties>
</ClassInfo>
</TypeInfo>
```

Customizing Apache Flex Scripts

You can manually customize your Flex scripts. You can insert verifications manually using the `Verify` function on Flex object properties. Each Flex object has a list of properties that you can verify. For a list of the properties available for verification, view a list of the supported Flex classes in the `com.borland.silktest.jtf.flex` package in the *API Reference*.

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

Testing Multiple Flex Applications on the Same Web Page

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

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

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



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

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



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

Adobe AIR Support

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

Silk Test provides a sample Adobe AIR application. You can access the sample application at <http://demo.borland.com/flex/SilkTest20.0/index.html> and then click the Adobe AIR application that you want to use. You can select the application with or without automation. In order to execute the AIR application, you must install the Adobe AIR Runtime.

Overview of the Flex Select Method Using Name or Index

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

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

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

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

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

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



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

Selecting an Item in the FlexDataGrid Control

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

1. To select an item in the FlexDataGrid control using the index value, use the `SelectIndex` method.

For example, type `FlexDataGrid.SelectIndex(1)`.

2. To select an item in the FlexDataGrid control using the content value, use the `Select` method.

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

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

Enabling Your Flex Application for Testing

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

- Apache Flex Automation Package
- Silk Test Automation Package

Apache Flex Automation Package

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

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



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

Silk Test Automation Package

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

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

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

Linking Automation Packages to Your Flex Application

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

When you embed functional testing classes in your application SWF file at compile time, the size of the SWF file increases. If the size of the SWF file is not important, use the same SWF file for functional testing

and deployment. If the size of the SWF file is important, generate two SWF files, one with functional testing classes embedded and one without. Use the SWF file that does not include the embedded testing classes for deployment.

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

- automation.swc
- automation_agent.swc
- FlexTechDomain.swc
- automation_charts.swc (include only if your application uses charts and Flex 2.0)
- automation_dmv.swc (include if your application uses charts and Flex > 3.x)
- automation_flasflexkit.swc (include if your application uses embedded flash content)
- automation_spark.swc (include if your application uses the new Flex 4.x controls)
- automation_air.swc (include if your application is an AIR application)
- automation_airspace.swc (include if your application is an AIR application and uses new Flex 4.x controls)

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

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



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

Precompiling the Flex Application for Testing

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

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

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



Note: If your application uses charts, you must also add the automation_charts.swc file.

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

The configuration files are located at:

Apache Flex 2 SDK – <flex_installation_directory>/frameworks/flex-config.xml

Apache Flex Data Services – <flex_installation_directory>/flex/WEB-INF/flex/flex-config.xml

The following example adds the automation.swc and automation_agent.swc files to the application:

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



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



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

Run-Time Loading

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

Limitations

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

```
import flash.display.*;

import flash.net.URLRequest;

import flash.system.ApplicationDomain;

import flash.system.LoaderContext;

var ldr:Loader = new Loader();

var urlReq:URLRequest = new URLRequest("RuntimeClasses.swf");
```

```
var context:LoaderContext = new LoaderContext();

context.applicationDomain = ApplicationDomain.currentDomain;

loader.load(request, context);
```

Loading at Run-Time

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

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

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

3. Add `file:///` as a prefix to the file path.
For example, if your file URL includes a parameter, such as: `?automationurl=explorer.swf`,
type: .

```
file:///C:/Program%20Files/Silk/Silk Test/ng/sampleapplications/Flex/3.2/
FlexControlExplorer32/FlexAutomationLauncher.html?automationurl=explorer.swf
```

Using the Command Line to Add Configuration Information

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

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

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



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

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

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

Passing Parameters into a Flex Application

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

Passing Parameters into a Flex Application Before Runtime

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

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

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

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

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

```
<script language="JavaScript" type="text/javascript">
    AC_FL_RunContent(eef
        "src", "FlexAutomationLauncher",
        "width", "100%",
        "height", "100%",
        "align", "middle",
        "id", "FlexAutomationLauncher",
        "quality", "high",
        "bgcolor", "white",
        "name", "FlexAutomationLauncher",
        "allowScriptAccess", "sameDomain",
        "type", "application/x-shockwave-flash",
        "pluginspage", "http://www.adobe.com/go/getflashplayer",
        "flashvars", "yourParameter=yourParameterValue"+
"&automationurl=YourApplication.swf"
    );
</script>
```

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

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

Creating Testable Flex Applications

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

- Providing Meaningful Identification of Objects
- Avoiding Duplication of Objects

Providing Meaningful Identification of Objects

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

To provide meaningful identification of objects:

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

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

Avoiding Duplication of Objects

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

To avoid duplicating objects:

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

Custom Attributes for Apache Flex Applications

Apache Flex applications use the predefined property `automationName` to specify a stable identifier for the Apache Flex control as follows:

```
<?xml version="1.0" encoding="utf-8"?>
  <s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300">
    <fx:Script>
      ...
    </fx:Script>
    <s:Button x="247" y="81" label="Button" id="button1" enabled="true"
click="button1_clickHandler(event)"
      automationName="AID_buttonRepeat"/>
    <s:Label x="128" y="123" width="315" height="18" id="label1"
verticalAlign="middle"
      text="awaiting your click" textAlign="center"/>
  </s:Group>
```

Apache Flex application locators look like the following:

```
...//SparkApplication//SparkButton[@caption='AID_buttonRepeat']
```



Attention: For Apache Flex applications, the `automationName` is always mapped to the locator attribute `caption` in Silk4J. If the `automationName` attribute is not specified, Silk4J maps the property `ID` to the locator attribute `caption`.

Flex AutomationName and AutomationIndex Properties

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

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



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

Flex Class Definition File

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

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

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

The XML file has the following basic structure:

```
<TypeInformation>
<ClassInfo>
<Implementation />
<Events>
<Event />
...
</Events>
<Properties>
<Property />
...
</Properties>
</ClassInfo>
</TypeInformation>
```

Setting the Flex automationName Property

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

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

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

Example

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

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

Example

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

```
<?xml version="1.0"?>
<!-- at/SimpleComboBox.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      [Bindable]
      public var cards: Array = [
        {label:"Visa", data:1},
        {label:"MasterCard", data:2},
        {label:"American Express", data:3}
      ];

      [Bindable]
      public var selectedItem:Object;
    ]
  >
</mx:Script>
<mx:Panel title="ComboBox Control Example">
  <mx:ComboBox id="cb1" dataProvider="{cards}"
    width="150"
    close="selectedItem=ComboBox(event.target).selectedItem"
    automationName="Credit Card List"
  />
  <mx:VBox width="250">
    <mx:Text width="200" color="blue" text="Select a type of
credit card." />
    <mx:Label text="You selected: {selectedItem.label}"/>
    <mx:Label text="Data: {selectedItem.data}"/>
  </mx:VBox>
</mx:Panel>
</mx:Application>
```

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

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

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

```
<mx:List xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:Script>
    <![CDATA[
      import mx.automation.IAutomationObject;
      override public function
      createAutomationIDPart(item:IAutomationObject):Object {
        var id:Object = super.createAutomationIDPart(item);
        id["automationName"] = id["automationIndex"];
        return id;
      }
    ]]>
  </mx:Script>
</mx:List>
```

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

Setting the Flex Select Method to Use Name or Index

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

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

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

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

2. Determine which XML file is related to the class that you want to modify.

The XML files related to the preceding controls include: `FlexCommonControls.xml`, `AdvancedDataGrid.xml`, or `OLAPDataGrid.xml`.

3. Navigate to the XML files that are related to the class that you want to modify.

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

4. Make the following adaptations in the corresponding XML file.

```
<ClassInfo Extends="FlexList" Name="FlexControlName"
EnableIndexedSelection="true" >
...
</ClassInfo>
```

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

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



Note: Setting the EnableIndexedSelection= to false in the code or removing the boolean returns recording to using the name (FlexList::Select events).

5. Re-start your Flex application and the Open Agent in order for these changes to become active.

Coding Flex Containers

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

Adding and Removing Containers from the Automation Hierarchy

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

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

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

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

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

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

```
</mx:HBox>
</mx:Panel>
</mx:Application>
```

Multiview Containers

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

Flex Automation Testing Workflow

The Silk4J workflow for testing Flex applications includes:

- Automated Testing Initialization
- Automated Testing Recording
- Automated Testing Playback

Flex Automated Testing Initialization

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

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

Flex Automated Testing Recording

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

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

7. FlexTechDomain uses the `AutomationManager.getParent()` method to get the logical parent of Button. The AutomationIDPart objects of parent controls are collected at each level up to the application level.
8. All the AutomationIDParts are included as part of the AutomationID object.
9. The FlexTechDomain sends the information in a call to Silk4J.
10. When the user stops recording, the `FlexTechDomain.endRecording()` method is called.

Flex Automated Testing Playback

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

1. For each script call, Silk4J contacts the Silk Test Agent and sends the information for the script call to be executed. This information includes the complete window declaration, the event name, and parameters.
2. The Silk Test Agent forwards that information to the FlexTechDomain.
3. The FlexTechDomain uses `AutomationManager.resolveIDToSingleObject` with the window declaration information. The AutomationManager returns the resolved object based on the descriptive information (automationName, automationIndex, id, and so on).
4. Once the Flex control is resolved, FlexTechDomain calls `AutomationManager.replayAutomatableEvent()` to replay the event.
5. The `AutomationManager.replayAutomatableEvent()` method invokes the `IAutomationObject.replayAutomatableEvent()` method on the delegate class. The delegate uses the `IAutomationObjectHelper.replayMouseEvent()` method (or one of the other replay methods, such as `replayKeyboardEvent()`) to play back the event.
6. If there are verifications in your script, FlexTechDomain invokes `AutomationManager.getProperties()` to access the values that must be verified.

Styles in Apache Flex Applications

For applications developed in Apache Flex 3.x, Silk4J does not distinguish between styles and properties. As a result, styles are exposed as properties. However, with Apache 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 Apache 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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards `?` and `*`.

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

Why Cannot Silk4J Recognize Apache Flex Controls?

If Silk4J cannot recognize the controls of an Apache Flex application, which you are accessing through a Web server, you can try the following things:

- Compile your Apache Flex application with the Adobe automation libraries and the appropriate `FlexTechDomain.swc` for the Apache Flex version.

- Use runtime loading.
- Apache Flex controls are not recognized when embedding an Apache Flex application with an empty `id` attribute.

Java AWT/Swing Support

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



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



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

Sample Applications

Silk Test provides a sample Swing test application. Download and install the sample applications from <http://supportline.microfocus.com/websync/SilkTest.aspx>. After you have installed the sample applications, click (in Microsoft Windows 7) **Start > Programs > Silk > Silk Test > Sample Applications > Java Swing > Swing Test Application** or (in Microsoft Windows 10) **Start > Silk**.

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported Controls

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

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

Attributes for Java AWT/Swing Applications

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

Supported attributes for Java AWT/Swing include:

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



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

Dynamically Invoking Java Methods

Dynamic invoke enables you to directly call methods, retrieve properties, or set properties, on an actual instance of a control in the application under test. You can also call methods and properties that are not

available in the Silk4J API for this control. Dynamic invoke is especially useful when you are working with custom controls, where the required functionality for interacting with the control is not exposed through the Silk4J API.

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

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

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

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

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



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



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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

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

- Enum types

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

- Lists

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

- Other controls

Control parameters can be passed or returned as `TestObject`.

Returned Values

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

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

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

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

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

1. If the test fails, because Silk4J cannot start the application, edit the application configuration.
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**.
 - b) In the **Executable Pattern** 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 Pattern** 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.

Oracle Forms Support

This functionality is supported only if you are using the Open Agent.

Silk4J provides built-in support for testing applications that are based on Oracle Forms.

 **Note:** For some controls, Silk4J provides only low-level recording support.

For information on the supported versions and browsers for Oracle Forms, refer to the [Release Notes](#). For a complete list of the controls available for Oracle Forms, view a list of the supported Oracle Forms classes in the API Reference.

Prerequisites for Testing Oracle Forms

To test an application that is built with Oracle Forms, the following prerequisites need to be fulfilled:

- The next-generation Java Plug-In needs to be enabled. This setting is enabled by default. You can change the setting in the **Java Control Panel**. For additional information on the next-generation Java Plug-In, refer to the Java documentation.
- To prevent Java security dialogs from displaying during a test run, the Applet needs to be signed.
- Micro Focus recommends that the Oracle Forms developer enables the `Names` property. When this property is enabled, the Oracle Forms runtime exposes the internal `name`, which is the name that the developer of the control has specified for the control, as the `Name` property of the control. Otherwise, Silk4J calculates a value for the Silk4J `Name` attribute, which usually consists of the class name of the control plus an index. This enables Silk4J to generate stable locators for controls.

Attributes for Oracle 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 Oracle Forms include:

- `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`

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

Java SWT and Eclipse RCP Support

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

Silk Test supports:

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported Controls

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

Java SWT Custom Attributes

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

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

Java SWT Example

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

```
Button myButton = Button(parent, SWT.NONE);  
  
myButton.setData("SilkTestAutomationId", "myButtonId");
```

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

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

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

Attributes for Java SWT Applications

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

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes



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

Dynamically Invoking Java Methods

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

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

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

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

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

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



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



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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

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

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

- Enum types

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

- Lists

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

- Other controls

Control parameters can be passed or returned as `TestObject`.

Returned Values

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

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

Troubleshooting Java SWT and Eclipse Applications

Some SWTTree methods do not replay with low-level playback

When using low-level playback, some SWTTree methods, for example `expand` and `collapse`, do not replay.

To solve this problem, set the replay mode to **Default**. For additional information, see *Setting Replay Options*.

Selecting a non-visible node in an SWTTree

When using low-level playback, Silk4J cannot interact with non-visible nodes in an SWTTree.

To solve this problem, set the replay mode to **Default**. For additional information, see *Setting Replay Options*.

Testing Mobile Applications

Silk4J enables you to automatically test your native mobile applications (apps) and mobile web applications. Automatically testing your mobile applications with Silk4J provides the following benefits:

- It can significantly reduce the testing time of your mobile applications.
- You can create your tests once and then test your mobile applications on a large number of different devices and platforms.
- You can ensure the reliability and performance that is required for enterprise mobile applications.
- It can increase the efficiency of QA team members and mobile application developers.
- Manual testing might not be efficient enough for an agile-focused development environment, given the large number of mobile devices and platforms on which a mobile application needs to function.

 **Note:** To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see *Licensing Information*.

 **Note:** Silk4J provides support for testing mobile apps on both Android and iOS devices.

For information on the supported operating system versions and the supported browsers for testing mobile applications, refer to the [Release Notes](#).

Android

Silk4J enables you to test a mobile application on an Android device or an Android emulator.

Prerequisites for Testing Mobile Applications on Android

Before you can test a mobile application (app) on an Android device or on an Android emulator, ensure that the following prerequisites are met:

- If you have created your own hybrid app by adding a web view to a native mobile app, add the following code to the app to make your app testable with Silk4J:

```
WebView.setWebContentsDebuggingEnabled(true);  
webView.getSettings().setJavaScriptEnabled(true);
```

- Silk4J does not support showing a live view of the device screen for Android 4.4. Micro Focus recommends using Android 5 or later.

Testing Mobile Applications on Android

To test a mobile application on a physical Android device or on an Android emulator, perform the following tasks:

1. Ensure that you have met the prerequisites for testing mobile applications on Android.
For additional information, see [Prerequisites for Testing Mobile Applications on Android](#).
2. If you want to test the mobile application on an Android emulator, configure the emulator settings for Silk4J.
For additional information, see [Configuring the Android Emulator for Silk Test](#).

3. Start the Android emulator or connect the device to the machine on which Silk4J is installed.
4. If you want to test the mobile application on a physical Android device that you are using for the first time on this machine, install the appropriate Android USB Driver on the machine.
For additional information, see [Installing a USB Driver](#).
5. If you want to test the mobile application on a physical Android device, enable USB-debugging on the Android device.
For additional information, see [Enabling USB-Debugging](#).
6. Create a Silk4J project for your mobile application.
7. Create a test for your mobile application.
8. Record the actions that you want to execute in the test. When you start the **Recording** window, the **Select Application** dialog box opens.
9. To test a mobile web application:
 - a) Select the **Web** tab.
 - b) Select the mobile browser that you want to use.
 - c) Specify the web page to open in the **Enter URL to navigate** text box.
10. To test a native mobile application or a Hybrid application:



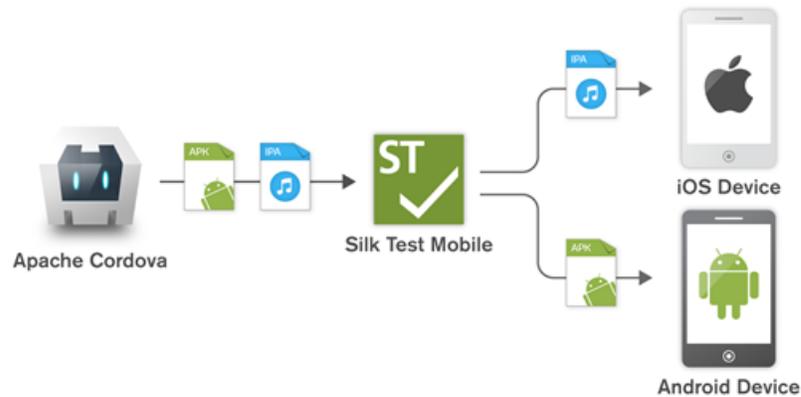
Note: To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see [Licensing Information](#).

- a) Select the **Mobile** tab.
 - b) Select the mobile device, on which you want to test the app, from the list.
 - c) Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
11. Click **Finish**.
An Android device or emulator must not be screen-locked during testing. To keep the device awake while it is connected to a machine, open the settings and tap **Developer Options**. Then check **Stay awake** or **Stay awake while charging**.
12. Use the **Recording** window to record the test against the mobile application.
For additional information, see [Recording Mobile Applications](#).
13. When you have recorded all the actions, stop the recording.
14. Replay the test.
15. Analyze the test results.

Testing Hybrid Applications on Android

Hybrid applications (apps) are apps that are run on the device, like native applications, but are written with web technologies, for example HTML5, CSS, and JavaScript.

Silk4J provides full browser support for testing debug hybrid apps that consist of a single web view, which is embedded in a native container. A common example of such a hybrid app would be an Apache Cordova application.



To prepare a non-debug hybrid app for testing, enable remote debugging in the app by adding the following code to the app:

```
WebView.setWebContentsDebuggingEnabled(true);
webView.getSettings().setJavaScriptEnabled(true);
```

To test non-debug hybrid apps without remote debugging enabled or hybrid apps that include more than one web view, enable the Silk4J fallback support by setting the option `OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT` to `TRUE`. For additional information, see *Setting Advanced Options*. With the fallback support enabled, Silk4J recognizes and handles the controls in a web view as native mobile controls instead of browser controls. For example, the following code clicks on a link when using browser support:

```
desktop.setOption(CommonOptions.OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT,
false);
desktop.<DomLink> find("//INPUT[@id='email']").click();
```

With the fallback support enabled, the following code clicks on the same link:

```
desktop.setOption(CommonOptions.OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT,
true);
desktop.<DomLink> find("//MobileTextField[@resource-id='email']").click();
```

Silk4J can detect web views that support Chrome remote debugging. Silk4J can detect web views with either the package `com.android.webview` or the package `com.google.android.webview`, which are the default packages on most Android devices.

 **Note:** Silk4J supports testing hybrid apps on Android 4.4 or later. To test hybrid apps on Android, Android System WebView version 51 or later is required.

The process for testing a hybrid app on Android is the same as the process for testing a mobile native application. For additional information, see [Testing Mobile Applications on Android](#).

Installing a USB Driver

To connect an Android device for the first time to your local machine to test your mobile applications, you need to install the appropriate USB driver.

The device manufacturer might provide an executable with all the necessary drivers for the device. In this case you can just install the executable on your local machine. If the manufacturer does not provide such an executable, you can install a single USB driver for the device on the machine.

To install the Android USB driver:

1. Download the appropriate driver for your device.

For example, for information on finding and installing a USB driver for a Google Nexus device, see <http://developer.android.com/tools/extras/oem-usb.html>.

2. Connect your Android device to a USB port on your local machine.
3. From your desktop or **Windows Explorer**, right-click **Computer** and select **Manage**.
4. In the left pane, select **Device Manager**.
5. In the right pane, locate and expand **Other device**.
6. Right-click the device name, for example *Nexus 5x*, and select **Update Driver Software**. The **Hardware Update Wizard** opens.
7. Select **Browse my computer for driver software** and click **Next**.
8. Click **Browse** and navigate to the folder to which you have downloaded the USB driver.
9. Select the USB driver.
10. Click **Next** to install the driver.

Enabling USB-Debugging

To communicate with an Android device over the Android Debug Bridge (adb), enable USB debugging on the device.

1. On the Android device, open the settings.
2. Tap **Developer Settings**.
The developer settings are hidden by default. If the developer settings are not included in the settings menu of the device:
 - a) Depending on whether the device is a phone or a pad, scroll down and tap **About phone** or **About Pad**.
 - b) Scroll down again and tap **Build Number** seven times.
3. In the **Developer settings** window, check **USB-Debugging**.
4. Set the USB mode of the device to **Media device (MTP)**, which is the default setting.
For additional information, refer to the documentation of the device.

Recommended Settings for Android Devices

To optimize testing with Silk4J, configure the following settings on the Android device that you want to test:

- Enable USB-debugging on the Android device. For additional information, see [Enabling USB-Debugging](#)
- An Android device must be connected as a media device to the machine on which the Open Agent is running. The USB mode of the Android device must be set to **Media device (MTP)**.
- An Android device or emulator must not be screen-locked during testing. To keep the device awake while it is connected to a machine, open the settings and tap **Developer Options**. Then check **Stay awake** or **Stay awake while charging**.

Configuring the Android Emulator for Silk4J



Note: When using an Android emulator, an additional adb server is running in addition to the one that is used by Silk4J. If the running adb servers have different versions, the connection between the Open Agent and the device might become unstable or even break. To avoid version mismatch errors, specify the path to the Android SDK directory by setting the environment variable `SILK_ANDROID_HOME`, for example to `C:\Users\<user>\AppData\Local\Android\android-sdk`. If the information service was running during this change, use the Windows Service Manager to restart the Silk Test information service with the updated environment variable. If the environment variable is not set, Silk4J uses the adb version that is shipped with Silk4J.

When you want to test mobile applications on an Android emulator with Silk4J, you have to configure the emulator for testing:

1. Install the latest version of the Android SDK.
For information on how to install and configure the Android SDK, see [Get the Android SDK](#).

2. Install Android Studio 2 or later.

 **Tip:** You can skip installing Android Studio and use the emulator provided with the Android SDK. However, Micro Focus recommends installing Android Studio for improved emulator performance. The remaining steps in this topic require Android Studio to be installed.

3. From Android Studio, start the **AVD Manager**.

4. Click **Create Virtual Device**.

5. Select a virtual device.

6. Click **Next**.

7. Download and select a system image of Android that includes Google APIs.

8. Click **Next**.

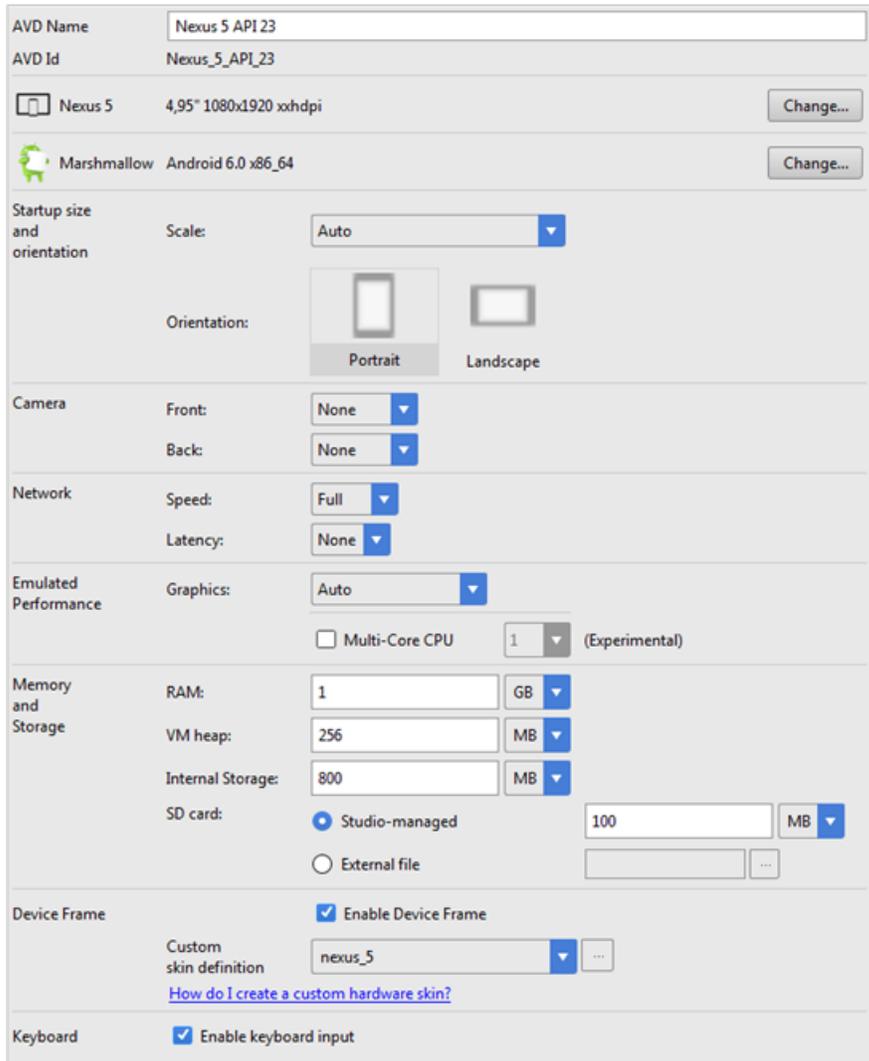
9. Configure the virtual device according to your requirements.

10. Click **Show Advanced Settings**.

11. Adjust the RAM size and the heap space used by the emulator to an amount that is manageable by your machine.

 **Tip:** Micro Focus recommends using at least 1 GB RAM and 256 MB heap space.

12. Select **Auto** from the list in the **Emulated Performance** area.



AVD Name: Nexus 5 API 23
AVD Id: Nexus_5_API_23

Nexus 5 4,95" 1080x1920 xxhdpi Change...

Marshmallow Android 6.0 x86_64 Change...

Startup size and orientation
Scale: Auto
Orientation: Portrait (selected), Landscape

Camera
Front: None
Back: None

Network
Speed: Full
Latency: None

Emulated Performance
Graphics: Auto
 Multi-Core CPU 1 (Experimental)

Memory and Storage
RAM: 1 GB
VM heap: 256 MB
Internal Storage: 800 MB
SD card: Studio-managed 100 MB
 External file

Device Frame
 Enable Device Frame
Custom skin definition: nexus_5
[How do I create a custom hardware skin?](#)

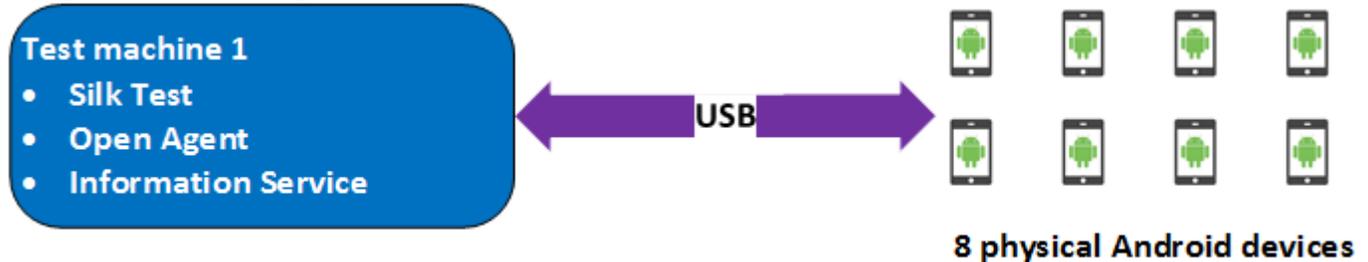
Keyboard
 Enable keyboard input

13. Click **Finish**.

Tested Configurations for Parallel Test Execution

With Silk4J, you can run automated tests on multiple Android devices in parallel. The amount of Android devices that you are able to use in parallel depends on the available hardware. Micro Focus has successfully tested the following hardware configurations:

Configuration with a single test machine

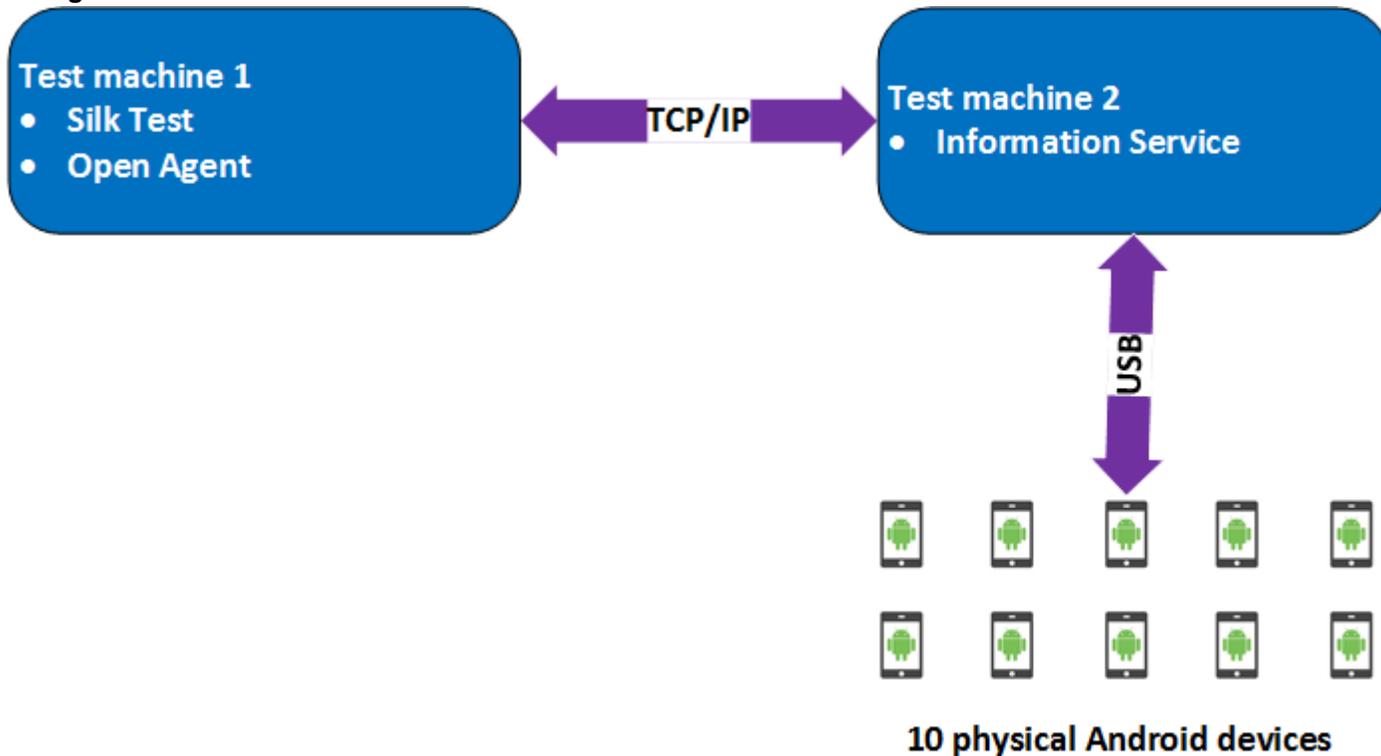


Using a single test machine directly connected to the Android devices through USB, we tested up to 8 physical Android devices in parallel.

The test machine was a Lenovo ThinkPad T450 with the following hardware specifications:

- Intel® Core™ i7 - 5600U CPU @ 2.60 GHz
- 2 cores (4 threads)
- 8 GB RAM

Configuration with two test machines



Here we are using two test machines, one with Silk4J installed and another, which is configured as a remote location for the first machine and has the Silk Test Information Service installed. Using such a configuration, we tested up to 10 physical Android devices in parallel.

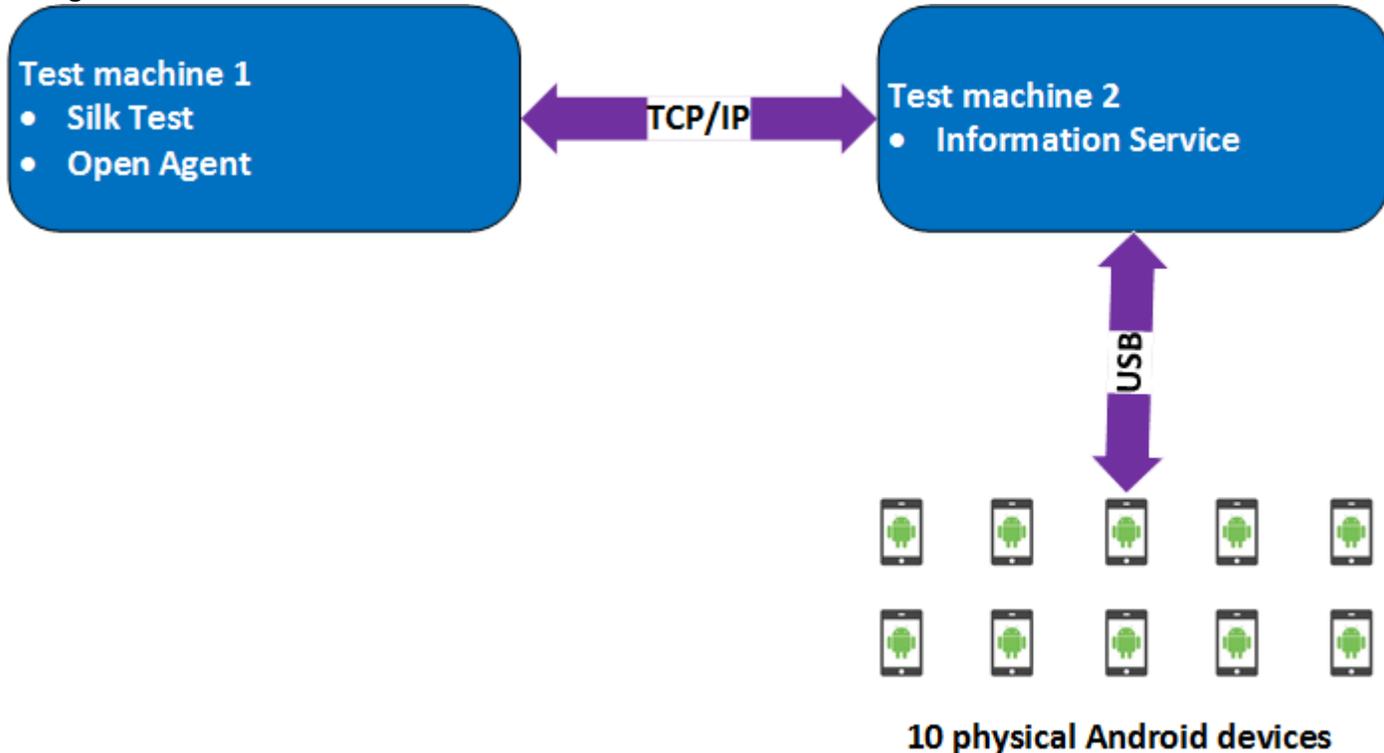
Test machine 1 was a Lenovo ThinkPad T450 with the following hardware specifications:

- Intel® Core™ i7 - 5600U CPU @ 2.60 GHz
- 2 cores (4 threads)
- 8 GB RAM

Test machine 2 was a Dell Precision T1700 with the following hardware specifications:

- Intel® Core™ i7 - 4770 CPU @ 3.40 GHz
- 4 cores (8 threads)
- 16 GB RAM

Configuration with a Windows machine and a Mac



Here we are using two test machines, a Windows machine with Silk4J installed and a Mac, which is configured as a remote location for the first machine and has the Silk Test Information Service installed. Using such a configuration, we tested up to 10 physical Android devices in parallel.

Test machine 1 was a Lenovo ThinkPad T450 with the following hardware specifications:

- Intel® Core™ i7 - 5600U CPU @ 2.60 GHz
- 2 cores (4 threads)
- 8 GB RAM

Test machine 2 was an Apple Mac Mini with the following hardware specifications:

- Intel® Core™ i5 - 4782U CPU @ 2.60 GHz
- 2 cores (4 threads)
- 16 GB RAM

iOS

Silk4J enables you to test a mobile application on an iOS device or an iOS Simulator.

Because of significant changes by Apple in iOS 9.3 in comparison to the previous versions of iOS, Silk Test supports testing mobile applications on iOS 9.3 or later. For a list of the supported iOS versions, refer to the [Release Notes](#).



Note: Testing mobile applications on iOS 11 requires Xcode 9. When using Xcode 9 on a Mac, testing on physical devices and Simulators with iOS versions prior to iOS 11 that are connected to or running on this Mac is not supported. Use Xcode 8.3 to test physical devices and Simulators with iOS 9.3 and iOS 10.

Prerequisites for Testing Mobile Applications on iOS

Before you can test a mobile application (app) on an iOS device or on an iOS Simulator, ensure that the following prerequisites are met:

- The current version of the Silk Test information service is installed on the Mac. For additional information, see [Installing the Silk Test Infoservice on a Mac](#).
- If you want to test your application on a physical iOS device, ensure the following:
 - The device is connected to the Mac.
 - The device has a supported version of iOS. For a list of the supported iOS versions, refer to the [Release Notes](#).
- If you want to test your application on an iOS Simulator, ensure the following:
 - The iOS Simulator image is installed on the Mac.
 - The iOS Simulator image has a supported version of iOS. For a list of the supported iOS versions, refer to the [Release Notes](#).
- If you want to test your application on a physical iOS device, ensure that the same time zone is set on the device and the Mac.
- A supported version of Xcode is installed on the Mac.
- Silk4J is installed on a Windows machine.
- The Mac is located in the same network as the Windows machine and is added as a remote location to the Windows machine.
- To test a native mobile app on an iOS device, ensure that the `.ipa` file of your app has been signed with a developer account. For additional information, see [Preparing an iOS App for Testing](#).
- To test a native mobile app on an iOS Simulator, ensure that the app has been zipped. For additional information, see [Testing Native Mobile Applications on an iOS Simulator](#).
- To test a native mobile app on both an iOS device and an iOS Simulator, ensure that the signed `.ipa` file and the zipped `.app` directory have the same name, except for the file extension, and are located in the same folder.
- To test a native mobile app, ensure that the ID of the iOS device is associated with the developer profile which was used to sign the app.
- The iOS device must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings > General > Passcode Lock**.
- The Mac should not switch off the screen during testing, otherwise the **Playback Status** dialog box will not display anything.
- To test a mobile application on an iOS Simulator, deactivate the display sleep on the Mac during testing.
- To test a native mobile app on a physical iOS device, enable the UI automation on the device. For additional information, see [Preparing an iOS Device for Testing](#).
- To test a mobile web application with Apple Safari on a physical iOS device, activate the **Web Inspector**. For additional information, see [Preparing an iOS Device for Testing](#).
- Micro Focus recommends using iOS devices which have a Lightning connector. Silk4J does not support showing a live view of the device screen for iOS devices that are not connected to a Mac through a Lightning cable.

Testing Native Mobile Applications on a Physical iOS Device

 **Note:** To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see [Licensing Information](#).

For information on the prerequisites for testing mobile applications on iOS, see [Prerequisites for Testing Mobile Applications on iOS](#). For information on the known limitations when testing native mobile applications, see [Limitations for Testing Mobile Native Applications](#).

To test a native mobile application (app) or a hybrid application on a physical iOS device, perform the following tasks:

1. Prepare the iOS device for testing.
For additional information, see [Preparing an iOS Device for Testing](#).
2. Prepare the app for testing.
For additional information, see [Preparing an iOS App for Testing](#).
3. Prepare the Mac for testing. For additional information, see [Preparing a Mac for Testing Mobile Applications on iOS](#).
4. Add the Mac, to which the iOS device is connected, as a remote location to the Windows machine on which Silk Test is installed.
For additional information, see [Editing Remote Locations](#).

 **Note:** At any given point in time, you can test on multiple physical iOS devices that are connected to the Mac, but only on one iOS Simulator that is running on the Mac. With Silk Test 17.5 Hotfix 1 or later, you are no longer required to use multiple user sessions on a Mac to test mobile applications on iOS.

5. Create a Silk4J project for your mobile application.
6. Create a test for your mobile application.
7. Record the actions that you want to execute in the test. When you start the **Recording** window, the **Select Application** dialog box opens.
8. Select the **Mobile** tab.
9. Select the mobile device, on which you want to test the app, from the list.
10. Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
11. Click **Finish**.

An iOS device or Simulator must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings > Touch ID & Code**.
12. When you have recorded all actions, stop recording.
13. Replay the test.
14. Analyze the test results.

 **Note:** To test a native mobile app on both an iOS device and an iOS Simulator, ensure that the signed `.ipa` file and the zipped `.app` directory have the same name, except for the file extension, and are located in the same folder.

Testing Native Mobile Applications on an iOS Simulator

 **Note:** To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see [Licensing Information](#).

For information on the prerequisites for testing mobile applications on iOS, see [Prerequisites for Testing Mobile Applications on iOS](#). For information on the known limitations when testing native mobile applications, see [Limitations for Testing Mobile Native Applications](#).

To test a native mobile application (app) or a hybrid application on an iOS Simulator, perform the following tasks:

1. Prepare the Mac for testing. For additional information, see [Preparing a Mac for Testing Mobile Applications on iOS](#).
2. In the Xcode project of the app, compile the app for the iOS Simulator.
You can compile the app either from the Xcode UI or from the command line. For example, to compile the app through the command line for an iOS Simulator with iOS 10.0, execute the following command:

```
xcodebuild -sdk iphonesimulator10.0
```
3. Zip up the `.app` directory of the app into a `.zip` file.
By default, the `.app` directory is located in the directory `~/Library/Developer/Xcode/DerivedData`. You can click **File > Project Settings** in Xcode to see into which location the directory is stored.
4. Add the Mac, on which the iOS Simulator is installed, as a remote location to the Windows machine on which Silk4J is installed.

For additional information, see [Editing Remote Locations](#).

 **Note:** You can only test on one iOS Simulator that is installed on a Mac. Multiple Silk4J users cannot simultaneously test on multiple iOS Simulators that are installed on the same Mac.

5. Create a Silk4J project for your mobile application.
6. Create a test for your mobile application.
7. Record the actions that you want to execute in the test. When you start the **Recording** window, the **Select Application** dialog box opens.
8. Select the **Mobile** tab.
9. Select the iOS Simulator from the list.
10. Select the native mobile application.
 - If you want to install the app on the iOS Simulator, click **Browse** to select the zipped app file or enter the full path to the zipped app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on the iOS Simulator, specify the **Bundle ID**.
11. Click **Finish**.
An iOS device or Simulator must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings > Touch ID & Code**.
12. When you have recorded all actions, stop recording.
13. Replay the test.
14. Analyze the test results.

 **Note:** To test a native mobile app on both an iOS device and an iOS Simulator, ensure that the signed `.ipa` file and the zipped `.app` directory have the same name, except for the file extension, and are located in the same folder.

Testing Mobile Web Applications on a Physical iOS Device

For information on the prerequisites for testing mobile applications on iOS, see [Prerequisites for Testing Mobile Applications on iOS](#). For information on the known limitations when testing mobile web applications, see [Limitations for Testing Mobile Web Applications](#).

To test a mobile web application on a physical iOS device, perform the following tasks:

1. Prepare the iOS device for testing.
For additional information, see [Preparing an iOS Device for Testing](#).
2. Prepare the Mac for testing. For additional information, see [Preparing a Mac for Testing Mobile Applications on iOS](#).
3. Add the Mac, to which the iOS device is connected, as a remote location to the Windows machine on which Silk Test is installed.
For additional information, see [Editing Remote Locations](#).



Note: At any given point in time, you can test on multiple physical iOS devices that are connected to the Mac, but only on one iOS Simulator that is running on the Mac. With Silk Test 17.5 Hotfix 1 or later, you are no longer required to use multiple user sessions on a Mac to test mobile applications on iOS.

4. Create a Silk4J project for your mobile application.
5. Create a test for your mobile application.
6. Record the actions that you want to execute in the test. When you start the **Recording** window, the **Select Application** dialog box opens.
7. To test a mobile web application:
 - a) Select the **Web** tab.
 - b) Select the mobile browser that you want to use.
 - c) Specify the web page to open in the **Enter URL to navigate** text box.
8. Click **Finish**.
An iOS device or Simulator must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings > Touch ID & Code**.
9. When you have recorded all actions, stop recording.
10. Replay the test.
11. Analyze the test results.

Testing Mobile Web Applications on an iOS Simulator

For information on the known limitations when testing mobile web applications, see [Limitations for Testing Mobile Web Applications](#).

To test a mobile web application on an iOS Simulator, perform the following tasks:

1. Prepare the Mac for testing. For additional information, see [Preparing a Mac for Testing Mobile Applications on iOS](#).
2. Add the Mac, on which the iOS Simulator is installed, as a remote location to the Windows machine on which Silk Test is installed.
For additional information, see [Editing Remote Locations](#).



Note: At any given point in time, you can test on multiple physical iOS devices that are connected to the Mac, but only on one iOS Simulator that is running on the Mac. With Silk Test 17.5 Hotfix 1 or later, you are no longer required to use multiple user sessions on a Mac to test mobile applications on iOS.

3. Create a Silk4J project for your mobile application.

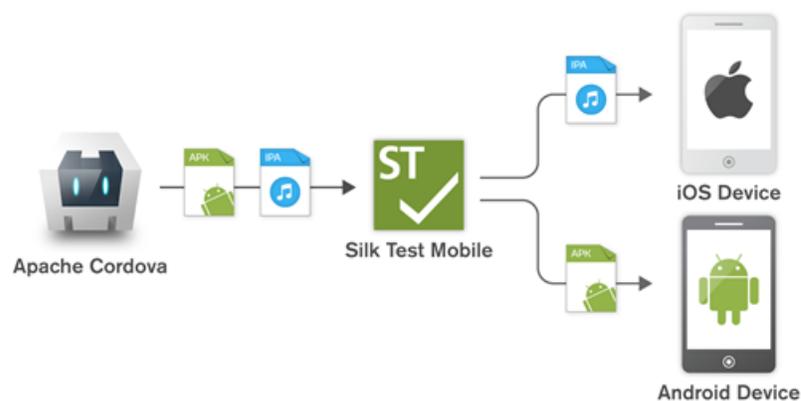
4. Create a test for your mobile application.
5. Record the actions that you want to execute in the test. When you start the **Recording** window, the **Select Application** dialog box opens.
6. To test a mobile web application:
 - a) Select the **Web** tab.
 - b) Select the mobile browser that you want to use.
 - c) Specify the web page to open in the **Enter URL to navigate** text box.
7. Click **Finish**.

An iOS device or Simulator must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings > Touch ID & Code**.
8. When you have recorded all actions, stop recording.
9. Replay the test.
10. Analyze the test results.

Testing Hybrid Applications on iOS

Hybrid applications (apps) are apps that are run on the device, like native applications, but are written with web technologies, for example HTML5, CSS, and JavaScript.

Silk4J provides full browser support for testing debug hybrid apps that consist of a single web view, which is embedded in a native container. A common example of such a hybrid app would be an Apache Cordova application.



To test non-debug hybrid apps without remote debugging enabled or hybrid apps that include more than one web view, enable the Silk4J fallback support by setting the option `OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT` to `TRUE`. For additional information, see *Setting Advanced Options*. With the fallback support enabled, Silk4J recognizes and handles the controls in a web view as native mobile controls instead of browser controls. For example, the following code clicks on a link when using browser support:

```
desktop.setOption(CommonOptions.OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT,
false);
desktop.<DomLink> find("//INPUT[@id='email']").click();
```

With the fallback support enabled, the following code clicks on the same link:

```
desktop.setOption(CommonOptions.OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT,
true);
desktop.<DomLink> find("//MobileTextField[@resource-id='email']").click();
```

The process for testing a hybrid app on iOS is the same as the process for testing a mobile native application. For additional information, see [Testing Native Mobile Applications on a Physical iOS Device](#) or [Testing Native Mobile Applications on an iOS Simulator](#).

Before testing a hybrid app on an iOS device, ensure that the **Web Inspector** is activated on the device. For additional information, see [Preparing an iOS Device for Testing](#).

Preparing an iOS Device for Testing



Note: To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see [Licensing Information](#).

To prepare the iOS device to test mobile applications:

1. Start Xcode on the Mac.
2. Connect the iOS device to the Mac.
3. On the iOS device, click **Settings > Developer**.



Tip: If the **Developer** menu is not displayed on the iOS device, restart the device and the Mac.

4. Activate **Enable UI Automation**.
5. To test a mobile web application on Apple Safari, click **Settings > Safari > Advanced**.
6. Activate the **Web Inspector**.
7. If you want to test on an iOS Simulator, enable **Rotate Device Automatically**.

You can enable this setting by using the **Silk Test Configuration Assistant** or by enabling it manually. To open the **Configuration Assistant** on a Mac, click on the Silk Test icon in the status menus and select **Configuration Assistant**. To enable the setting manually, perform the following actions:

- a) On the Mac, start the iOS Simulator.
- b) With Xcode 9 or later, expand the **Hardware** menu.
With prior versions of Xcode, expand the **Debug** menu.
- c) Check **Rotate Device Automatically**.

Preparing an iOS App for Testing

To be able to test a specific iOS app on a specific iOS device with Silk4J, consider the following:

- Test automation is only possible with iOS apps that can be installed manually on specific iOS devices. To be able to sign an iOS app, you require a membership in the *Apple Developer Program*. For additional information, see [Choosing a Membership](#). To test without having a membership in the *Apple Developer Program*, see [Using a Personal Team Profile for Testing on Physical iOS Devices](#).



Note: You cannot automatically test iOS apps that are created for publication in the App Store, as well as apps that can be installed manually on any iOS device.

- Before you can install and execute an iOS app on a specific iOS device, you have to register the iOS device with your Apple Developer account.
- You have to use Xcode to create an IPA file of the iOS app, which you can then install on the iOS device. To create IPA files for testing on a specific iOS device, members of the Apple Developer Program can use the *Archive* mechanism of Xcode, by using one of the following two options:
 - If you are a member of the *Apple Developer Enterprise Program*, you can use the **Save for Ad Hoc Deployment** option.
 - If you are a member of the Apple Developer Program, but not of the Apple Developer Enterprise Program, you can use the **Save for Development Deployment** option.

For additional information, see [Exporting Your App for Testing \(iOS, tvOS, watchOS\)](#).

To be able to test a specific iOS app on an iOS Simulator with Silk4J, use Xcode to create a ZIP file of the iOS app, which you can then install on the iOS Simulator. For additional information, refer to the Xcode documentation.

Installing the Silk Test Information Service on a Mac

 **Note:** To install the information service on a Mac, you require administrative privileges on the Mac.

To create and execute tests on a Mac using Apple Safari or using iOS or Android devices, install the Silk Test information service (information service) on the Mac. Once the information service is installed and active, you can record and replay tests from a Silk4J client that is installed on a Windows machine.

 **Note:** You cannot record on a Mac. To add a Mac as a test location to Silk4J, see *Editing Remote Locations* in the Silk4J documentation. .

To install the information service on a Mac:

1. Read the information in the topic *Prerequisites for Testing with Apple Safari on a Mac* in the Silk4J documentation.
2. Ensure that a Java JDK is installed on the Mac.
3. If you want to test mobile applications on an iOS device, ensure that Xcode is installed on the Mac.
4. Access the information service setup file, `SilkTestInformationService<Version>-<Build Number>.pkg`.
 - If you have downloaded the information service setup file while installing Silk Test, open the folder `macOS` in the Silk Test installation directory, for example `C:\Program Files (x86)\Silk\SilkTest`.
 - If you have not downloaded the information service setup file while installing Silk Test, you can download the setup file from [Micro Focus SupportLine](#).
5. Copy the file `SilkTestInformationService<Version>-<Build Number>.pkg` to the Mac.
6. Execute `SilkTestInformationService<Version>-<Build Number>.pkg` to install the information service.
7. Follow the instructions in the installation wizard.
8. When asked for the password, provide the password of the currently signed in Mac user.
9. When Apple Safari opens and a message box asks whether to trust the SafariDriver, click **Trust**.

 **Note:** If you want to test against Apple Safari 10 or prior on macOS 10.12 (Sierra) or prior, SafariDriver needs to be installed on the Mac. You can only install the SafariDriver if you are directly logged in to the Mac, and not connected through a remote connection.
10. To complete the installation, the installer logs the current Mac user out. To verify that the information service was installed correctly, log in to the Mac.
11. If you are installing the information service on a Mac with macOS Mojave (10.14) or later, you might have to enable additional automation permissions for Silk Test after logging in to the Mac. If permissions need to be granted, Silk Test will automatically show request permission dialogs.
 - a) Click **OK** to acknowledge the information dialog.
 - b) Click **OK** in all sub-sequent request permission dialogs.

 **Important:** If you do not enable these permissions for Silk Test, you will not be able to test web applications against Google Chrome or mobile applications on an iOS device or on a Simulator on this Mac. If by mistake you have clicked **Don't Allow** in one of the permission dialogs, open a terminal on the Mac and type the following command:

```
sudo tccutil reset AppleEvents
```

Then restart the Mac and accept the permission dialogs by clicking **OK**.

12. Click on the Silk Test icon in the top-right corner of the screen to see the available devices and browsers.

 **Tip:** If the Silk Test icon does not appear, restart the Mac.

Preparing a Mac to Test Mobile Applications on iOS

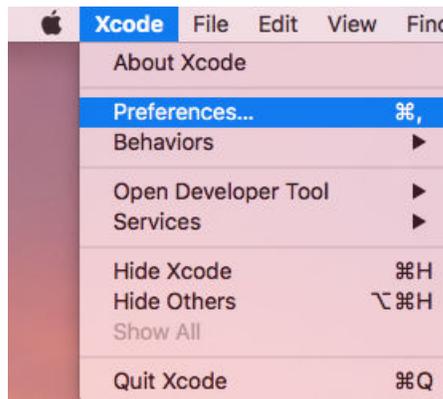
 **Note:** To test native mobile applications or hybrid applications with Silk4J, you require a native mobile license. For additional information, see [Licensing Information](#).

To test mobile applications on iOS, you require a Mac to which you can connect the iOS device, or on which the iOS Simulator is running. This Mac requires Xcode to be installed. For additional information on the prerequisites for testing mobile applications on iOS, see [Prerequisites for Testing Mobile Applications on iOS](#).

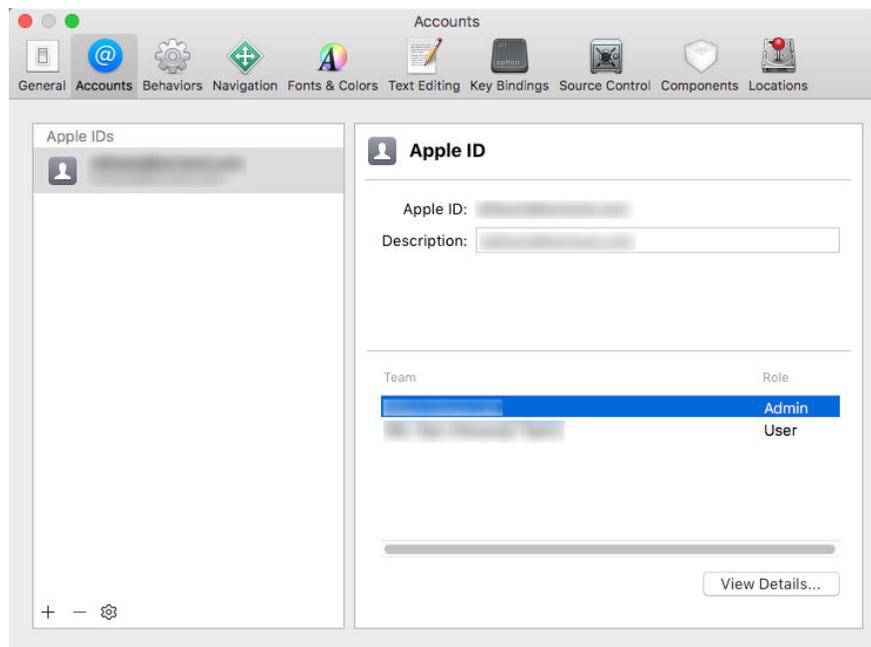
To execute iOS tests on a physical iOS device, follow the instructions in the **Silk Test Configuration Assistant** to configure the `WebDriverAgentRunner` Xcode project. To open the **Configuration Assistant**, click on the Silk Test icon in the status menus and select **Configuration Assistant**.

If for any reason you want to manually build the `WebDriverAgentRunner` Xcode project, perform the following actions:

1. Start Xcode on the Mac.
2. Select **Xcode > Preferences**.



3. In the **Preferences** window, select your account.
 - a) Select the **Accounts** tab.
 - b) Choose your **Apple ID**.
 - c) Choose your **Team**.
 - d) Click **View Details**.



4. Access the **Apple Member Center** and retrieve your development team.
5. In a terminal, navigate to `~/ .silk/silktest/conf/`.
6. Rename the xcconfig file template `silktest.xcconfig.sample` to `silktest.xcconfig`.
7. Add your development team to the `silktest.xcconfig` file.

```
DEVELOPMENT_TEAM = <your development team>
```

8. Execute the following commands in a terminal on the Mac to verify that you have prepared the `WebDriverAgentRunner` project correctly:

- a) Determine the unique device id (`udid`) of your physical iOS device:

```
iddevice_id -l
```

- b) Navigate to the `WebDriverAgentRunner` project:

```
cd /Application/Silk/Mobile/common/Appium/node_modules/appium-xcuitest-driver/WebDriverAgent
```

- c) Test that the `WebDriverAgent` can be built:

```
xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -xcconfig ~/.silk/silktest/conf/silktest.xcconfig -destination 'id=<udid>' test
```

Replace the `<udid>` with the unique device id that you have determined previously.



Tip: If the `xcodebuild` command fails, follow the instructions in the error message. Additionally, open the Preferences window of the `WebDriverAgentRunner` project and ensure that the **Automatically manage signing** check box in the **General** tab is not checked.

9. *Optional:* In the `infoservice.properties` file, you can specify the port for the Silk Test Information Service or capabilities which are used during all test runs on the Mac.

For additional information, see [Editing the Properties of the Silk Test Information Service](#).

Using a Personal Team Profile for Testing on Physical iOS Devices

If you have no membership in the Apple Developer Program, you can use a personal team profile to test an application on a physical iOS device:

1. On the Mac, navigate to `/Application/Silk/Mobile/common/Appium/node_modules/appium-xcuitestdriver/WebDriverAgent`.

2. Open `WebDriverAgent.xcodeproj` project in Xcode.
3. From the **TARGETS** list, select the `WebDriverAgentLib` target:
 - a) Click the **General** tab.
 - b) Select **Automatically manage signing**.
 - c) Select your development team.

The **Signing Certificate** is automatically selected.
4. From the **TARGETS** list, select the `WebDriverAgentRunner` target:
 - a) Click the **General** tab.
 - b) Select **Automatically manage signing**.
 - c) Select your development team.

The **Signing Certificate** is automatically selected.
5. If Xcode fails to create a provisioning profile for the `WebDriverAgentRunner` target, manually change the bundle id for the target.
 - a) Click the **Build Settings** tab.
 - b) Change the **Product Bundle Identifier** to something that Xcode accepts.

For example, if the **Product Bundle Identifier** is `com.facebook.WebDriverAgentRunner`, change it to `io.appium.WebDriverAgentRunner` or `io.borland.WebDriverAgentRunner`.
 - c) Click the **General** tab.

The target should now have a provisioning profile.
6. Save the `WebDriverAgent.xcodeproj` project.
7. To verify that everything works as expected, open a terminal and build the project:


```
xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination 'id=<udid>' test IPHONEOS_DEPLOYMENT_TARGET=10.3
```
8. To avoid problems during the reinstallation of the `WebDriverAgent` apps, permanently install an additional app that uses the same provisioning profile, on the device. For example, install the `IntegrationApp` of the `WebDriverAgent` Xcode project:
 - a) From the **TARGETS** list, select the `IntegrationApp` target.
 - b) Click the **General** tab.
 - c) Select **Automatically manage signing**.
 - d) Select your development team.
9. If Xcode fails to create a provisioning profile for the `IntegrationApp` target, manually change the bundle id for the target in the same way as described above for the `WebDriverAgentRunner` target.
10. After successfully configuring the `IntegrationApp` target, install and run the `IntegrationApp` on the physical iOS device:
 - a) Select the target and the iOS device.
 - b) Click **Play**.

Although the apps are successfully installed on the device, an error message like the following might appear in the console or the Appium log files:

```
2017-01-24 09:02:18.358 xcodebuild[30385:339674] Error Domain=com.apple.platform.iphoneros
Code=-12 "Unable to launch com.apple.test.WebDriverAgentRunner-Runner"
UserInfo={NSLocalizedString=Unable to launch com.apple.test.WebDriverAgentRunner-Runner,
NSUnderlyingError=0x7fa839cad60 {Error Domain=DTXMessage Code=1 "(null)"
UserInfo={DTXExceptionKey=The operation couldn't be completed. Unable to launch
com.apple.test.WebDriverAgentRunner-Runner because it has an invalid code signature, inadequate
entitlements or its profile has not been explicitly trusted by the user. : Failed to launch process with bundle
identifier 'com.apple.test.WebDriverAgentRunner-Runner'}}} 2017-01-24 09:02:18.358
xcodebuild[30385:339674] Error Domain=IDETestOperationsObserverErrorDomain Code=5 "Early
unexpected exit, operation never finished bootstrapping - no restart will be attempted"
UserInfo={NSLocalizedString=Early unexpected exit, operation never finished bootstrapping - no
```

restart will be attempted} Testing failed: Test target WebDriverAgentRunner encountered an error (Early unexpected exit, operation never finished bootstrapping - no restart will be attempted)
The problem is that the developer is not trusted on the device. If you manually try to run the apps on the device, you will see an **Untrusted Developer** message.

To solve this issue on the device, go to **Settings > General > Profiles** or **Settings > General > Device Management**, depending on the device type and the iOS version. Then trust the developer and allow the apps to be run.

Editing the Properties of the Silk Test Information Service

Use the `infoservice.properties` file to specify the port for the Silk Test Information Service, whether to use a secure connection through HTTPS, or the capabilities that are applied each time Silk Test executes a test on the machine on which the Silk Test Information Service is running.

1. Navigate to the directory in which the `infoservice.properties.sample` file is located.
 - On a Windows machine, navigate to `%PROGRAMDATA%\Silk\SilkTest\conf`, for example `C:\ProgramData\Silk\SilkTest\conf`.
 - On macOS, navigate to `~/ .silk/silktest/conf/`.
2. Rename the file `infoservice.properties.sample` to `infoservice.properties`.
3. Specify whether Silk4J should communicate with the information service over a secure connection through HTTPS.
 - To use a secure connection through HTTPS, set `infoservice.https.enabled` to `true`. This is the default setting.
 - To disable using a secure connection through HTTPS, set `infoservice.https.enabled` to `false`.
4. *Optional:* If you have specified that you want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.https.port`.
The default HTTPS port is 48561. Port numbers can be any number from 1 to 65535.
5. *Optional:* To redirect all HTTP requests to the HTTPS port, if you have specified that you want to use a secure connection through HTTPS, set `infoservice.http-to-https.enabled` to `true`.
The default value is `false`.
6. *Optional:* If you have specified that you do not want to use a secure connection through HTTPS, you can specify a different port that is not in use through which Silk4J should communicate with the information service as the `infoservice.default.port`.
The default port is 22901.
7. *Optional:* To replace the certificates that are used by Silk Test for the HTTPS connection with your own certificates, see [Replacing the Certificates that are Used for the HTTPS Connection to the Information Service](#).
8. To specify capabilities, add the following line to the `infoservice.properties` file:

```
customCapabilities=<custom_capability_1>;<custom_capability_2>;...
```

Example: Running an iOS Simulator in a Specified Language

To always run a specific iOS Simulator on a Mac in the same language, for example Japanese, specify the custom capabilities *language* and *locale*. To do so, add the following line to the `infoservice.properties` file:

```
customCapabilities=language=ja;locale=ja_JP
```

Uninstalling the Silk Test Information Service from a Mac

To uninstall the Silk Test information service (information service) from a Mac, for example if you no longer want to execute tests against Apple Safari on the Mac:

1. On the Mac, create a new shell file, for example `uninstallInfoService.sh`.
2. Type the following code into the new file:

```
#!/bin/sh

if launchctl list | grep com.borland.infoservice ; then
    launchctl unload /Library/LaunchAgents/com.borland.infoservice.plist
    echo "unloading Launch Daemon"
fi

if [ -d "/Applications/Silk" ]
then
    sudo rm -rf /Applications/Silk
fi

if [ -f "/Library/LaunchAgents/com.borland.infoservice.plist" ]
then
    sudo rm /Library/LaunchAgents/com.borland.infoservice.plist
fi

if [ -f "/usr/local/bin/ideviceinstaller" ]
then
    sudo rm /usr/local/bin/ideviceinstaller
fi

exit 0
```

3. In the command line, type `chmod +x uninstallInfoService.sh` to make the shell file executable.
4. Execute the shell file from the command line.

Recommended Settings for iOS Devices

To optimize testing with Silk4J, configure the following settings on the iOS device that you want to test:

- To make the testing reflect the actions an actual user would perform, disable AutoFill and remembering passwords for Apple Safari. Tap **Settings** > **Safari** > **Passwords & AutoFill** and turn off the **Names and Passwords** setting.
- The iOS device must not fall into sleep mode during testing. To turn the screen lock and password off, select **Settings** > **General** > **Passcode Lock**.

Running Existing Scripts on iOS Using XCUITest

 **Attention:** Prior Silk4J versions used *Instruments* to automate iOS devices. With iOS 9.3, Apple has replaced the support for Instruments with support for the *XCUITest* framework, causing Silk4J to also no longer support *Instruments*. Because of this change, existing iOS test scripts might break when executed from the current version of Silk4J.

- The behavior of the `classname` attribute in XCUITest is different to the behavior in Instruments. In most cases, Silk4J will automatically handle this change. However, if an existing test script breaks because of such a `classname` attribute, you will have to record a new locator for the corresponding object.
- The object hierarchy has changed.

Recording Mobile Applications

Once you have established the connection between Silk4J and a mobile device or an emulator, you can record the actions that are performed on the device. To record mobile applications, Silk4J uses a **Recording** window that provides the following functionality:

- Displays the screen of the mobile device or Android emulator which you are testing.
- When you perform an action in the **Recording** window, the same action is performed on the mobile device.
- When you interact with a control on the screen, the **Recording** window preselects the default action.
 - If the default action is a `Click`, and you left-click on the control, the action is executed. You can perform a right-click to show a list of the available actions against the control. You can then select the action that you want to perform and click **OK**.
 - If the default action is not a `Click`, a list of all the available actions against the control displays, and you can select the action that you want to perform or simply accept the preselected action by clicking **OK**.

When you have selected an action from the list, you can type values for the parameters of the selected action into the parameter fields. Silk4J automatically validates the parameters.

- During recording, Silk4J displays the mouse position next to the recording window. You can toggle the location to switch between displaying the absolute mouse position on the device display and the mouse position in relation to the active object.
- When you pause the recording, you can perform actions in the screen which are not recorded to bring the device into a state from which you want to continue recording.
- When you stop recording, a script is generated with your recorded actions, and you can proceed with replaying the test.

Selecting the Mobile Device for Test Replay

You can define the mobile device that is used for the replay of a test in the following ways:

- If you execute a test from the UI of Silk4J and the **Select Mobile Device** dialog box displays, the mobile device, Android Emulator, or iOS Simulator that is selected in the dialog box is used, and Silk4J ignores which mobile device is set in the test script.
- If the **Select Mobile Device** dialog box is disabled, because the **Don't show again** check box is checked, the application configurations in the individual test scripts determine the mobile device that is used to execute the tests.



Note: To re-enable the **Select Mobile Device** dialog box, click **Silk4J > Edit Application Configurations** and check the **Show 'Select Mobile Device' dialog before record and playback** check box

- If you execute a script from the command line or from a Continuous Integration (CI) server, specify the connection string in the application configuration of the script.

To overwrite the mobile device that is specified in the application configuration, use the `silktest.configurationName` environment variable.

- If you execute a test from Silk Central, specify the mobile device in the **Mobile Device Selection** area of the **Deployment** tab of the execution definition in Silk Central instead of specifying a connection string. For additional information, refer to the [Silk Central Help](#).

You can use the connection string to specify a specific mobile device, or you can filter a subset of the available devices, for example if you have a device pool. The first matching device is used for replay. If not specified otherwise, mobile devices are matched by using the following rules, with declining priority:

- Matching mobile devices connected to the local machine are preferred over mobile devices connected to remote locations.

- If the browser type is specified in the connection string, newer browser versions are preferred over older versions.
- Newer platforms are preferred over older platforms.
- A physical device is preferred to an Emulator or Simulator.
- A device with a device name that is alphabetically later is preferred. For example, a device named "iphone 6" is preferred to a device named "iphone 5".

Example: Connection string for an app on an Android device that is connected to a remote machine

To test the app `MyApp.apk` on an Android device that is connected to a remote machine, the connection string would look like the following:

```
"platformName=Android;deviceName=MotoG3;host=http://10.0.0.1;app=MyApp.apk"
```

Example: Connection string for an app on an iOS Simulator on a Mac

```
"platformName=iOS;platformVersion=10.0;deviceName=iPhone6;host=10.0.0.1;app=MyApp.ipa;isSimulator=true"
```

Using Devices from Mobile Center Directly from Silk4J

Micro Focus Mobile Center (Mobile Center) provides an end-to-end quality lab of real devices and emulators that help you test, monitor, and optimize your mobile apps for an enhanced user experience.

 **Note:** Silk4J supports testing against devices that are managed by Mobile Center 3.0 or later.

You can either access the devices that are managed by Mobile Center directly from Silk4J, or through Silk Central.

To access the devices that are managed by Mobile Center directly from Silk4J:

1. Add Mobile Center as a remote location.
For additional information, see [Editing Remote Locations](#).
2. To test on iOS, ensure that the following IPA files are signed:
 - HP4M-Agent.ipa
 - HPMC-AgentLauncher.ipa
 - WebDriverAgentRunner-Runner.ipa

 **Note:** Silk4J does not support testing iOS simulators through Mobile Center.

In the **Select Applications** dialog, you can now select the Mobile Center device on which you want to test.

 **Note:** You cannot test the same device with Silk Test Mobile and Mobile Center at the same time. When switching between these products, you need to remove all apps that are used for automation and restart the mobile device.

Android Before testing an Android device with Silk Test Mobile or before installing Mobile Center, you need to remove the following apps from the device:

- Appium Android Input Manager
- Appium Settings
- io.appium.uiautomator2.server
- io.appium.uiautomator2.server.text

- MC Agent
- Silk Screencast
- Unlock

iOS Before testing an iOS device with Silk Test Mobile or before installing Mobile Center, you need to remove the WebDriverAgent from the device.

Mobile Center might re-install some of these apps.

If you want to replay tests from a CI server or tests from the command line on a device that is managed by Mobile Center, you can specify the connection string for the device instead of configuring the remote connection. The connection string should look like the following:

```
"deviceName=MotoG3;platformName=Android;host=http://<Mobile Center server>:8080;hostType=MC;userName=<Mobile Center user name>;password=<Mobile Center password>"
```

 **Note:** For security reasons, Silk4J saves an encrypted form of the Mobile Center password when you create a new application configuration. Micro Focus recommends using the encrypted password in the connection string.

 **Note:** When testing on a device that is managed through the Mobile Center, Silk4J does not support using the methods `typeKeys` or `setText` to type key codes like **ENTER**. Additionally, Silk4J does not support pressing the **Home** button on iOS devices.

 **Note:** When testing on an Android Emulator, disable the GPU HW Acceleration.

Using Devices from Mobile Center through Silk Central

Micro Focus Mobile Center (Mobile Center) provides an end-to-end quality lab of real devices and emulators that help you test, monitor, and optimize your mobile apps for an enhanced user experience.

 **Note:** Silk4J supports testing against devices that are managed by Mobile Center 3.0 or later.

You can either access the devices that are managed by Mobile Center directly from Silk4J, or through Silk Central.

To access the devices that are managed by Mobile Center through Silk Central:

1. Integrate Silk4J with Silk Central.

For additional information, see *Integrating Silk4J with Silk Central*.

2. Configure Silk Central to use Mobile Center.

 **Note:** While installing Mobile Center, ensure that the appropriate Android SDK version is used. Ensure that the same version is used in Silk4J by setting the environment variable `SILK_ANDROID_HOME`, for example to `C:\Users\\AppData\Local\Android\android-sdk`. For additional information, refer to the *Silk Central Help*.

3. To test on iOS, ensure that the following IPA files are signed:

- HP4M-Agent.ipa
- HPMC-AgentLauncher.ipa
- WebDriverAgentRunner-Runner.ipa

 **Note:** Silk4J does not support testing iOS simulators through Mobile Center.

In the **Select Applications** dialog, you can now select the Mobile Center device on which you want to test.

 **Note:** You cannot test the same device with Silk Test Mobile and Mobile Center at the same time. When switching between these products, you need to remove all apps that are used for automation and restart the mobile device.

Android Before testing an Android device with Silk Test Mobile or before installing Mobile Center, you need to remove the following apps from the device:

- Appium Android Input Manager
- Appium Settings
- io.appium.uiautomator2.server
- io.appium.uiautomator2.server.text
- MC Agent
- Silk Screencast
- Unlock

iOS Before testing an iOS device with Silk Test Mobile or before installing Mobile Center, you need to remove the WebDriverAgent from the device.

Mobile Center might re-install some of these apps.

 **Note:** When testing on a device that is managed through the Mobile Center, Silk4J does not support using the methods `typeKeys` or `setText` to type key codes like **ENTER**. Additionally, Silk4J does not support pressing the **Home** button on iOS devices.

 **Note:** When testing on an Android Emulator, disable the GPU HW Acceleration.

Installing the Certificate for an HTTPS Connection to Mobile Center

To use a secure connection through HTTPS between Silk4J and Mobile Center, install the certificate for Mobile Center into the installation directory of the Java that is used by the Open Agent.

1. Open a browser.
2. Navigate to the HTTPS URL of the secure Mobile Center server.
3. Export the certificate of the secure Mobile Center server.

For information about exporting the certificate, refer to the documentation of the browser that you are using.

For example, if you are using Mozilla Firefox, perform the following actions:

- a) When Mozilla Firefox informs you that the connection is not secure, click **Advanced**.
 - b) Click **Add Exception...**
 - c) In the **Add Security Exception** dialog, click **View**.
 - d) In the **Certificate Viewer**, select the **Details** tab.
 - e) Click **Export**.
 - f) In the **Save Certificate To File** dialog, browse to the location in which you want to store the certificate file.
 - g) Specify a name for the certificate, for example `mc.crt`.
 - h) Click **Save**.
4. Browse to the location into which you have downloaded the certificate file.
 5. Double-click the certificate file and install the certificate using the wizard.
 - a) Select **Place all certificates in the following store**, and then click **Browse**.
 - b) Select **Trusted Root Certification Authorities**, and then complete the wizard.
 6. If a security warning appears, click **Yes** to confirm the installation.

7. Restart the Open Agent.
8. Refresh the Mobile Center server or connector machine URL in the browser and verify that there are no SSL certificate-related errors..

Changing the Mobile Center Password

When your Mobile Center password changes, use the **Edit Location** dialog to apply the new password in Silk4J.

1. Click **Silk4J > Edit Remote Locations**. The **Remote Locations** dialog box appears.
2. Select the Mobile Center remote location and click **Edit**. The **Edit Location - Mobile Center** dialog appears.
3. Type the new password into the **Password** field.
4. *Optional:* Click **Test** to test if the new password works.
5. If you are using the same Mobile Center in one or more connection strings, click **Copy** to copy the password in encrypted form to the clipboard. You can then paste the copied encrypted password into the connection strings.
6. Click **OK**.

Using SauceLabs Devices

SauceLabs provides an automated testing platform, enabling you to test on various mobile devices and mobile platform versions without having to purchase and maintain your own infrastructure.

To access SauceLabs devices through Silk Central, perform the following actions:

1. Ensure that Silk4J is integrated with Silk Central.
For additional information, see *Integrating Silk4J with Silk Central*.
2. Ensure that Silk Central is configured to use SauceLabs.
For additional information, refer to the *Silk Central Help*.

In the **Select Applications** dialog, you can now select the SauceLabs device on which you want to test.

Connection String for a Mobile Device

The *connection string* specifies which mobile device is used for testing. When performing mobile testing, Silk4J uses the connection string to connect to the mobile device. The connection string is typically part of the application configuration. You can set the connection string when you configure your application under test. To change the connection string, you can use the **Edit Application Configuration** dialog box.



Note: If you execute a test from Silk Central, specify the mobile device in the **Mobile Device Selection** area of the **Deployment** tab of the execution definition in Silk Central instead of specifying a connection string. For additional information, refer to the [Silk Central Help](#).

You can use the connection string to specify a specific mobile device, or you can filter a subset of the available devices, for example if you have a device pool. The first matching device is used for replay. If not specified otherwise, mobile devices are matched by using the following rules, with declining priority:

- Matching mobile devices connected to the local machine are preferred over mobile devices connected to remote locations.
- If the browser type is specified in the connection string, newer browser versions are preferred over older versions.
- Newer platforms are preferred over older platforms.
- A physical device is preferred to an Emulator or Simulator.
- A device with a device name that is alphabetically later is preferred. For example, a device named "iphone 6" is preferred to a device named "iphone 5".

The following components are available for the connection string:

Component	Description
deviceName	The name of the mobile device. When testing on a physical mobile device, the device ID can be used instead. Supports wildcards. Case-insensitive.
platformName	Android or iOS. Required.
deviceId	<i>Optional:</i> The ID of the mobile device. Can be used instead of the device name, when testing on a physical mobile device. Supports wildcards. Case-insensitive.
platformVersion	<i>Optional:</i> The Android or iOS version. Specify the version to test only on mobile devices that have a specific Android or iOS version. Supports wildcards. Case-insensitive.
browserVersion	<i>Optional:</i> Can be used in combination with the browser type to test only on the specified browser version. Supports wildcards. Case-insensitive.
host	<i>Optional:</i> If not set, any specified remote location can be used as the host. Supports wildcards. Case-insensitive.
app	The full path to an app that is not installed on the mobile device yet. For example <code>app=MyApp.apk</code> .
<ul style="list-style-type: none"> appPackage appActivity 	To test an app that is already installed on an Android device, specify both the package of the app and the activity that you want to use. For example <code>appPackage=silktest.insurancemobile;appActivity=.LoginActivity</code> .
bundleId	The identifier of the bundle of an app that is already installed on an iOS device. For example <code>app=MyApp.ipa</code> or <code>bundleId=silktest.InsuranceMobile</code> .
mobileCenterAppIdentifier	The identifier of an app that is already installed on a mobile device that is managed by Mobile Center. For example <code>mobileCenterAppIdentifier=com.microfocus.silktest.testapp</code> .
mobileCenterAppUploadNumber	The upload number of an app that is already installed on a mobile device that is managed by Mobile Center. Can be used to identify the application uniquely when the application is uploaded more than once. If this number is not specified, Silk4J uses the latest upload. For example <code>mobileCenterAppUploadNumber=3</code> .
noReset	<i>Optional:</i> Can be set when testing native mobile applications. Is only valid if the <i>app</i> is specified. True if the app should not be reinstalled before testing. False if the app should be reinstalled before testing. The default value is False.
isSimulator	<i>Optional:</i> Used to specify that the test should only be executed on an iOS Simulator. The device name can be used instead.
isPhysicalDevice	<i>Optional:</i> Used to specify that the test should only be executed on a physical device. The device name can be used instead.

When using a pool of devices and to find out which device is actually used for testing, you can use the return value of the `generateConnectionString` method of `MobileDevice` class.

Testing a mobile web application on a mobile device or on an Android Emulator

When testing a mobile web application on a mobile device or on an Android Emulator, the connection string consists of the following parts:

1. The mobile device name, for example `MotoG3`, or the device ID, for example `11111111`.



Note: If the device name is unique, Micro Focus recommends to use the device name in the connection string, because the device ID is less readable.

2. The platform name.
3. The browser version. This can only be used in combination with setting the browser type
4. The IP address or the host name of a specific remote machine, for example `10.0.0.1`. You can also use the name of a remote location that is specified in the **Edit Remote Locations** dialog box as the host name, for example `MyRemoteLocation`. When using the remote location name, you can also use

wildcards. To test an Android device that is connected to the local machine, specify the IP address or the host name of the local machine.

Example: Connection string for any available Android device

```
"platformName=Android"
```

Example: Connection string for a browser on an Android device that is connected to the local machine

To test a mobile browser on an Android device that is connected to the local machine, the connection string should look similar to the following:

```
"deviceName=MotoG3;platformName=Android;host=localhost"
```

or

```
"platformName=Android;deviceId=11111111;host=localhost"
```

Example: Connection string for a browser on an Android device that is connected to a remote machine

To test a mobile browser on a remote Android device, the connection string should look similar to the following:

```
"deviceName=MotoG3;platformName=Android;host=10.0.0.1"
```

```
"deviceName=MotoG3;platformName=Android;host=MyRemoteLocation*"
```

Example: Connection string for an Android device that is managed by Mobile Center

If you want to replay tests from a CI server or tests from the command line on a device that is managed by Mobile Center, you can specify the connection string for the device instead of configuring the remote connection. The connection string should look like the following:

```
"deviceName=MotoG3;platformName=Android;host=http://<Mobile Center server>:8080;hostType=MC;userName=<Mobile Center user name>;password=<Mobile Center password>"
```



Note: For security reasons, Silk4J saves an encrypted form of the Mobile Center password when you create a new application configuration. Micro Focus recommends using the encrypted password in the connection string.

Example: Connection string for a browser on an iOS device that is connected to a Mac

To test a mobile browser on a remote iOS device, the connection string would look like the following:

```
"deviceName=myiPhone6;platformName=iOS;host=10.0.0.1"
```

Testing a native mobile application on a mobile device or on an Android Emulator

When testing a native mobile application on a mobile device or on an Android Emulator, the connection string consists of the following parts:

1. The mobile device name, for example MotoG3, or the device ID, for example 11111111.



Note: If the device name is unique, Micro Focus recommends to use the device name in the connection string, because the device ID is less readable.

2. The platform name.
3. The IP address or the host name of a specific remote machine, for example 10.0.0.1. You can also use the name of a remote location that is specified in the **Edit Remote Locations** dialog box as the host name, for example *MyRemoteLocation*. When using the remote location name, you can also use wildcards. To test an Android device that is connected to the local machine, specify the IP address or the host name of the local machine.
4. The name of the file of the app that you want to test, or the URL of the file, if the file is located on a web server. For example `C:/MyApp.apk` or `MyApp.ipa`.
 - Android apps are always `.apk` files.
 - iOS apps on a real device are always `.ipa` files.
 - iOS apps on a Simulator are either a zipped file or a directory with the name `app`.

Example: Connection string for an app on an Android device that is connected to a remote machine

To test the app `MyApp.apk` on an Android device that is connected to a remote machine, the connection string would look like the following:

```
"platformName=Android;deviceName=MotoG3;host=http://10.0.0.1;app=MyApp.apk"
```

Example: Connection string for an app on an iOS device that is connected to a Mac

To test the app `MyApp.ipa` on an iOS device that is connected to a remote machine, the connection string would look like the following:

```
"platformName=iOS;deviceName=MyiPhone;host=http://10.0.0.1;app=MyApp.ipa"
```

Testing a mobile web application on an iOS Simulator

When testing a mobile web application on an iOS Simulator, the connection string consists of the following parts:

1. The platform name, which is `iOS`.
2. The platform version, for example `10.0`.
3. The mobile device name, for example `iPhone6`.
4. The IP address or the host name of the Mac, on which the iOS Simulator is running.

Example: Connection string for a browser on an iOS Simulator on a Mac

```
"platformName=iOS;platformVersion=10.0;deviceName=iPhone6;host=10.0.0.1;isSimulator=true"
```

Testing a native mobile application on an iOS Simulator

When testing a native mobile application on an iOS Simulator on a Mac, the connection string consists of the following parts:

1. The platform name, which is `iOS`.
2. The platform version, for example `10.0`.

3. The mobile device name, for example iPhone6.
4. The IP address or the host name of the remote machine, for example 10.0.0.1.
5. The name of the app that you want to test, for example MyApp.ipa.

Example: Connection string for an app on an iOS Simulator on a Mac

```
"platformName=iOS;platformVersion=10.0;deviceName=iPhone6;host=10.0.0.1;app=MyApp.ipa;isSimulator=true"
```

Interacting with a Mobile Device

To interact with a mobile device and to perform an action like a swipe in the application under test:

1. In the **Recording** window, click **Show Mobile Device Actions**. All the actions that you can perform against the mobile device are listed.
2. Select the action that you want to perform from the list.
3. To record a swipe on an Android device or emulator, move the mouse while holding down the left mouse button.
4. Continue with the recording of your test.

Releasing a Mobile Device

When recording or playing back a test against a mobile device, the Open Agent instance takes ownership of the device. By doing so, the Open Agent is preventing other Silk Test users from using the device. To enable other Silk Test users to use the device after you have finished recording or replaying tests on the device, Silk Test automatically releases the device when the Silk Test client is closed, when an unattended test process finishes, or when the Open Agent is closed. You can also manually release the device.



Note: Releasing a mobile device will close the application under test (AUT) on the mobile device.

Releasing a Mobile Device After Recording

Release a mobile device after recording to enable other Silk Test users to test on the device.

To release a mobile device after you have finished recording, perform one of the following actions:

- Stop the Open Agent from the System Tray.
- Close Silk4J. The device is only released by this action when parallel testing is enabled.
- If you are testing a device that is managed by Mobile Center, you can also release the device through the Mobile Center UI. For additional information, refer to the Mobile Center documentation.



Note: Releasing a mobile device will close the application under test (AUT) on the mobile device.

Releasing a Mobile Device After Replay

Release a mobile device after replay to enable other Silk Test users to test on the device.

To manually release a mobile device after replaying is complete, you can also perform one of the following:

- If you have tested a mobile web application, use the `close` method or the `closeSynchron` method of the `BrowserApplication` class. For additional information on these methods, refer to the API documentation.

```
webBrowser.close();
```

- If you have tested a mobile native application, use the `closeApp` method of the `MobileDevice` class.

For example, type the following:

```
MobileDevice mobileDevice = desktop.find("//MobileDevice");
mobileDevice.closeApp();
```

- Add the `desktop.detachAll()` statement to the test script.

A mobile device is automatically released if one of the following conditions is met:

- The Open Agent is closed.
- The test process stops during unattended testing. The device is only released by this action when parallel testing is enabled.
- Silk4J is closed. The device is only released by this action when parallel testing is enabled.



Note: Releasing a mobile device will close the application under test (AUT) on the mobile device.

Using the setLocation Method when Testing a Mobile Application

- When testing a native mobile application on iOS, the `setLocation` method is not supported.
- When testing a native mobile application on an Android version prior to Android 6.0, you have to enable **Allow mock locations** to use the `setLocation` method. To do so, open the settings of the Android device or emulator and tap **Developer Options**.
- When testing a native mobile application on Android 6.0 or later, you have to set the app to **Appium Settings** to use the `setLocation` method. To do so, open the settings of the Android device or emulator and tap **Developer Options > Select mock location app**. Then choose **Appium Settings**.



Note: The **Appium Settings** entry is only available if you have already executed a test with Appium on the Android device or emulator.

- To use the `setLocation` method when testing on an Android emulator on a Windows machine:
 1. On the Windows machine on which the emulator is running, navigate to the `%userprofile%` folder.
 2. Remove the content of the file `.emulator_console_auth_token`. This file is created when the emulator is started for the first time.
 3. Set the `.emulator_console_auth_token` file to be read only.

Troubleshooting when Testing Mobile Applications

Why does the Select Application dialog not display my mobile devices?

If Silk4J does not recognize a mobile device or emulator, the **Mobile** tab in the **Select Application** dialog does not display the device or emulator. Additionally, the **Web** tab of the **Select Application** dialog does not display the mobile browsers that are installed on the device or emulator.

Silk4J might not recognize a mobile device or emulator for one of the following reasons:

Reason	Solution
The emulator is not running.	Start the emulator.
The Android Debug Bridge (adb) does not recognize the mobile device.	To check if the mobile device is recognized by adb: <ol style="list-style-type: none">1. Navigate to the Android Debug Bridge (adb) in the Android SDK installation folder. If the Android SDK is not installed, navigate to <code>C:\Program Files (x86)\Silk\SilkTest\ng\Mobile\windows\AndroidTools\platform-tools</code> to use the adb that is installed with Silk4J.

Reason	Solution
	<ol style="list-style-type: none"> 2. Hold Shift and right-click into the File Explorer window. 3. Select Open command window here. 4. In the command window, type <code>adb devices</code> to get a list of all attached devices. 5. If your device is not listed, check if USB-debugging is enabled on the device and if the appropriate USB driver is installed. 6. If you get an error, for example <code>adb server is out of date</code>, ensure that the adb version in <code>C:\Program Files (x86)\Silk\SilkTest\ng\Mobile\windows\AndroidTools\platform-tools</code> is the same as the adb version of your local Android SDK. For additional information, see <i>What can I do if the connection between the Open Agent and my device is unstable?</i>
The version of the operating system of the device is not supported by Silk4J.	For information on the supported mobile operating system versions, refer to the Release Notes .
The USB driver for the device is not installed on the local machine.	Install the USB driver for the device on the local machine. For additional information, see <i>Installing a USB Driver</i> .
USB-debugging is not enabled on the device.	Enable USB-debugging on the device. For additional information, see <i>Enabling USB-Debugging</i> .



Note: If all previous solutions do not work, you could try to restart the device.

Why does Silk4J search for a URL in Chrome for Android instead of navigating to the URL?

Chrome for Android might in some cases interpret typing an URL into the address bar as a search. As a workaround you can manually add a command to your script to navigate to the URL.

What do I do if the adb server does not start correctly?

When the Android Debug Bridge (adb) server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients. All adb clients use port 5037 to communicate with the adb server. The adb server locates emulator and device instances by scanning odd-numbered ports in the range 5555 to 5585, which is the range used by emulators and devices. Adb does not allow changing those ports. If you encounter a problem while starting adb, check if one of the ports in this range is already in use by another program.

For additional information, see <http://developer.android.com/tools/help/adb.html>.

What can I do if the connection between the Open Agent and my device is unstable?

If you have installed the Android SDK or another tool that uses the Android Debug Bridge (adb), an additional adb server might be running in addition to the one that is used by Silk4J. If the running adb servers have different versions, the connection between the Open Agent and the device might become unstable or even break.

To avoid version mismatch errors, specify the path to the Android SDK directory by setting the environment variable `SILK_ANDROID_HOME`, for example to `C:\Users\\AppData\Local\Android\android-sdk`. If the information service was running during this change, use the Windows Service

Manager to restart the Silk Test information service with the updated environment variable. If the variable is not set, Silk4J uses the adb version that is shipped with Silk4J.

Why do I get the error: Failed to allocate memory: 8?

This error displays if you are trying to start up the emulator and the system cannot allocate enough memory. You can try the following:

1. Lower the RAM size in the memory options of the emulator.
2. Lower the RAM size of Intel HAXM. To lower the RAM size, run the `IntelHaxm.exe` again and choose **change**.
3. Open the **Task Manager** and check if there is enough free memory available. If not, try to free up additional memory by closing a few programs.

Why do I get the error "Silk Test cannot start the app that you have specified" during testing on an iOS device?

This error might display for one or more of the following reasons:

Reason	Solution
The iOS device is not in developer mode.	<p>You can enable the developer mode in one of the following two ways:</p> <ul style="list-style-type: none"> • Connect the device to a Mac on which Xcode is installed, and start the app that you want to test on the device. • Add your provisioning profiles to the device. <ol style="list-style-type: none"> 1. Open Xcode. 2. Select Window > Devices. 3. Right-click on the iOS device. 4. Select Show Provisioning Profiles. 5. Add your provisioning profiles.
You have recently updated the iOS version of the device.	<ol style="list-style-type: none"> 1. Open Xcode. 2. Select Window > Devices. 3. Wait until Xcode has processed the symbol files.
UI automation is not enabled on the iOS device.	<ol style="list-style-type: none"> 1. Select Settings > Developer. 2. Activate Enable UI Automation.
The Web Inspector is not activated on the iOS device, while you are trying to test a mobile web application.	<ol style="list-style-type: none"> 1. Click Settings > Safari > Advanced. 2. Activate the Web Inspector.
The app that you want to test was not built for the iOS version of the iOS device on which you are testing.	<p>Use Xcode to build the app for the iOS version of the device.</p>
The Software Update dialog box is currently open on the iOS device.	<p>Close the dialog box and disable automatic software updates:</p> <ol style="list-style-type: none"> 1. Select Settings > App and iTunes Stores > AUTOMATIC DOWNLOADS. 2. Deactivate Updates.

Why does my Android device display only the Back button in the dynamic hardware controls?

If the Android or the Android Emulator is screen-locked when you start testing, the device or Emulator might display only the button **Back** in the dynamic hardware controls.

To solve this issue, stop the Open Agent, restart the device, and change the device settings to no longer lock the screen.

Why does my Android device or emulator no longer display a keyboard?

To support unicode characters, Silk4J replaces the standard keyboard with a custom keyboard. When testing is finished, the original keyboard is restored. If an error occurs during testing, the custom keyboard might still be active and cannot be replaced.

To solve this issue, manually reset the keyboard under **Settings > Language & input > Current Keyboard**.

Why does my device not respond during testing?

If the device, emulator, or Simulator is screen-locked when you start testing, and Silk4J is unable to unlock the screen, the device, emulator, or Simulator might stop responding to any actions.

To solve this issue, stop the Open Agent and change the device settings to no longer lock the screen.

Why can I not install the Information Service on a Mac?

When the **Allow apps downloaded from** setting in the **General** tab of the **Security & Privacy** system preferences pane is set to **Mac App Store and identified developers**, which is the default value, the following error message appears when opening the Information Service setup:

"SilkTestInformationService<version>.pkg" can't be opened because it is from an unidentified developer.

To solve this issue, use one of the following:

- Right-click the setup file and select **Open**. A warning message will appear, but you will still be able to open the file.
- Set the **Allow apps downloaded from** setting to **Anywhere**.
- After attempting to open the file, navigate to the **General** tab of the **Security & Privacy** system preferences pane and click **Open Anyway**.

Why is the Recording window black when recording an Android app?

Android apps that require a higher level of security, for example apps that handle financial transactions, might have the [FLAG_SECURE](#) flag set, which prevents Silk4J from capturing the app. Silk4J relies on screenshots or on a video of the Android device during recording and will display a black screen of the device in the **Recording** window, if the Android app that you are testing has this flag set. To test such an app with Silk Test, you have to contact the app development team, and ask them to un-set the `FLAG_SECURE` flag during testing.

Why does Silk4J not show a video when testing on an Android emulator?

If the emulator is using the graphic card of your computer for better rendering, the video capturing of Silk4J might not work. To solve this, emulate the graphics in software:

1. Open the **Android Virtual Device Manager**.
2. Click **Edit** in the **Actions** column of the emulator.
3. Select **Software** from the list in the **Emulated Performance** area of the **Virtual Device Configuration** dialog.

What can I do if Silk4J does not show a video when testing in a cloud environment?

When testing in a cloud environment, showing a video might not work when recording or replaying a test, for example because required ports are not open.

To solve this issue, you can specify a list of WebDriver host URLs in the `infoservice.properties` file. For information on how to access this properties file, see *Editing the Properties of the Silk Test Information Service*. Add the option `infoservice.disableScreencastHosts` to the file, by typing the following:

```
infoservice.disableScreencastHosts=<URL_1>,<URL_2> , ...
```

For example:

```
infoservice.disableScreencastHosts=http://my-webdriver-server-url.com:80/wd/hub
```

You can specify URL patterns like `*my-webdriver-server-url.com` by using asterisks (*) as wildcards.

Silk4J will show a series of screenshots instead of a video when recording and replaying on the specified hosts.

How can I change the installed version of Xcode?

If the version of Xcode that you are using is not supported by Silk4J, for example when you upgrade to the latest version of Xcode, an error message might appear when testing on iOS.

To replace the installed version of Xcode with a supported version, download a supported Xcode version from <https://developer.apple.com/download/more/>, and replace the unsupported version with the downloaded version. For information about the supported Xcode versions, refer to the [Release Notes](#).

What can I do if my Mac runs out of disk space?

Silk4J uses Instruments to automate iOS devices. This tool creates large log files in the `/Library/Caches/com.apple.dt.instruments` directory, which might fill up disk space on the Mac. To solve this issue, Micro Focus recommends regularly deleting these log files, either manually or by using a cronjob. For example, to delete the files each day at the same time, you could do the following:

1. Type `sudo crontab -e` into a Terminal. This opens an editor in which you can edit the crontab for root.
2. Add the following line to the crontab:

```
0 2 1 * * find /Library/Caches/com.apple.dt.instruments -mtime +10 -delete
```
3. Save the crontab.

In this example, all log files that are older than ten days will be deleted each day at 2 AM from the directory.

Why does my test fail with the error message "Unable to sign WebDriver Agent for testing"?

When testing on a physical iOS device, this error usually means that during the build process the `WebDriverAgent` app could not be signed or that there is a problem with the provisioning profile.

You can check the actual problem with the following commands, which have to be executed at the Mac machine to which the device is connected:

```
cd /Applications/Silk/Mobile/common/Appium/node_modules/appium-xcuitest-driver
xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -
destination 'id=<udid>' test
```

Verify that the folder `Resources` exists under `/Applications/Silk/Mobile/common/Appium/node_modules/appium-xcuitest-driver` and that the folder contains the file `WebDriverAgent.bundle`. If not, create this folder and an empty `WebDriverAgent.bundle` file, for example by using the following command:

```
mkdir -p Resources/WebDriverAgent.bundle
```

What can I do to prevent the Developer Tools Access from requesting to take control of another process?

When starting the execution of a test on iOS, a message box stating the following might appear: Developer Tools Access needs to take control of another process for debugging to continue. Type your password to allow this.

To avoid getting this message, execute the following command in a Terminal:

```
sudo /usr/sbin/DevToolsSecurity --enable
```

Why are the rectangles wrong while testing a mobile web application on an iPad?

If the rectangles around controls are offset when testing a mobile web application on an iPad, you might have multiple browser tabs open and the Tab bar might be displayed. To fix this issue, close all tabs except one.

Why can I no longer record or replay tests on my device after updating Silk4J?

When updating to a new version of Silk4J, some Appium apps on any physical mobile devices that have already been used for mobile testing with the previous version of Silk4J are updated automatically. If for any reason these apps are not automatically updated, you might experience difficulties when trying to record or replay tests on the device.

If you are experiencing such issues on a specific Android device after updating Silk4J, manually uninstall the following apps from the device:

- Appium Android Input Manager
- Appium Settings
- io.appium.uiautomator2.server
- io.appium.uiautomator2.server.test
- Unlock

If you are experiencing such issues on a specific iOS device after updating Silk4J, manually uninstall the WebDriverAgentRunner from the device.

Why can I not record a mobile application?

Silk4J uses Appium to test mobile applications. Some network proxy settings set in Appium might interfere with recording Silk4J. You could try to deactivate the network proxy settings on the mobile device or Emulator.

Why can I not test on my Android device?

Some Android devices might have additional settings that prevent Silk4J from testing mobile applications on the device. For example, the Xiaomi Mi Mix 2 has the same prerequisites for testing as every other device, but these are not enough. To prepare the Xiaomi Mi Mix 2 for testing, perform the following actions:

1. Enable the developer mode on the device.
2. Navigate to **Settings > Additional settings > Developer options**.
3. Enable **USB debugging**.
4. Enable **Stay awake**.
5. Enable **Install via USB**.
6. Enable **USB debugging (Security settings)**.
7. Disable **Turn on MIUI optimizations**.

How Can I Use Chrome for Android to Replay Tests?

By default you can use the **Select Browser** dialog box to select the browser during replay.

If you execute a script from the command line or from a Continuous Integration (CI) server, you can specify the connection string in the application configuration of the script. To overwrite the browser that is specified in the application configuration, use the `silktest.configurationName` environment variable.

You can also use the property `browserType` of the `BrowserApplication` class to set the type of the browser that is used during replay. However, the `browserType` does not include an explicit value for Chrome for Android.

To specify that you want to use Chrome for Android as the browser, on which a test is replayed, set the `browserType` to `GoogleChrome` and specify `Android` as the platform. When `Android` is specified, Silk4J uses Chrome for Android instead of Google Chrome on a desktop machine to execute the test.

Examples

The following code sample shows how you can set the base state for a test to use Chrome for Android on a Nexus 7 by using the `silktest.configurationName`:

```
SET
silktest.configurationName="platformName=Android;deviceName=Nexus
7;host=10.0.0.1 - Chrome"
```

The following code sample shows how you can set the base state for a test to use Chrome for Android by using the `browserType`:

```
BrowserBaseState baseState = new
BrowserBaseState(BrowserType.GoogleChrome, "demo.borland.com/
InsuranceWebExtJS/");
baseState.setConnectionString("platformName=Android");
baseState.execute(desktop);
```

Limitations for Testing Mobile Web Applications

The support for playing back tests and recording locators on mobile browsers is not as complete as the support for the other supported browsers. The known limitations for playing back tests and recording locators for mobile web applications are:

- The following classes, interfaces, methods, and properties are currently not supported for mobile web applications:
 - `BrowserApplication` class.
 - `closeOtherTabs` method
 - `closeTab` method
 - `existsTab` method
 - `getActiveTab` method
 - `getSelectedTab` method
 - `getSelectedTabIndex` method
 - `getSelectedTabName` method
 - `getTabCount` method
 - `imageClick` method
 - `openTab` method
 - `selectTab` method
 - `DomElement` class.
 - `domDoubleClick` method
 - `domMouseMove` method
 - `getDomAttributeList` method

- `IKeyable` interface.
 - `pressKeys` method
 - `releaseKeys` method
- Silk4J does not support testing HTML frames and iFrames with Apple Safari on iOS, including text recognition in HTML frames and iFrames.

Text recognition includes the following methods:

- `textCapture`
- `textClick`
- `textExists`
- `textRectangle`
- Recording in landscape mode is not supported for emulators that include virtual buttons in the system bar. Such emulators do not correctly detect rotation and render the system bar in landscape mode to the right of the screen, instead of the lower part of the screen. However, you can record against such an emulator in portrait mode.
- Only HTML attributes in the HTML DOM are supported in XPath expressions for mobile applications. Silk4J does not support properties in XPath expressions.
- If you are testing a mobile web application on Android, Silk4J does not support zooming.
- The following JavaScript alert-handling methods of the `BrowserWindow` class do not work when testing on the Original Android Stock (AOSP) Browser:
 - `acceptAlert` method
 - `dismissAlert` method
 - `getAlertText` method
 - `isAlertPresent` method
- At any given point in time, you can test on multiple physical iOS devices that are connected to the Mac, but only on one iOS Simulator that is running on the Mac.
- Before starting to test a mobile web application, ensure that no browser tab is open.
 -  **Tip:** On iPads you can disable tabs in Apple Safari. Navigate to **Settings > Safari** and disable **Show Tab Bar** to do so.
- While testing a mobile web application, you can only have one browser tab open.
- Silk4J does not support testing mobile web applications that are opened by a native mobile application.

Limitations for Testing Native Mobile Applications

The known limitations for playing back tests and recording locators on native mobile applications (apps) are:

- The following classes, interfaces, methods, and properties are currently not supported for native mobile applications:
 - `IKeyable` interface.
 - `pressKeys` method
 - `releaseKeys` method
 - `MobileDevice` class.
 - When testing a native mobile application on iOS, the `setLocation` method is not supported.
 - When testing a native mobile application on an Android version prior to Android 6.0, you have to enable **Allow mock locations** to use the `setLocation` method. To do so, open the settings of the Android device or emulator and tap **Developer Options**.
 - When testing a native mobile application on Android 6.0 or later, you have to set the app to **Appium Settings** to use the `setLocation` method. To do so, open the settings of the Android

device or emulator and tap **Developer Options > Select mock location app**. Then choose **Appium Settings**.



Note: The **Appium Settings** entry is only available if you have already executed a test with Appium on the Android device or emulator.

- When testing on iOS, the `getValue` method of the `XCUIElementTypeSwitch` class returns the strings `false` or `true` depending on the checked state, instead of returning the strings `0` and `1`.
- Recording in landscape mode is not supported for Android emulators that include virtual buttons in the system bar. Such emulators do not correctly detect rotation and render the system bar in landscape mode to the right of the screen, instead of the lower part of the screen. However, you can record against such an emulator in portrait mode.
- Only HTML attributes in the HTML DOM are supported in XPath expressions for mobile applications. Silk4J does not support properties in XPath expressions.
- At any given point in time, you can test on multiple physical iOS devices that are connected to the Mac, but only on one iOS Simulator that is running on the Mac. With Silk Test 17.5 Hotfix 1 or later, you are no longer required to use multiple user sessions on a Mac to test mobile applications on iOS.
- Silk4J does not support text recognition when testing native mobile applications on both Android and iOS.

Text recognition includes the following methods:

- `textCapture`
- `textClick`
- `textExists`
- `textRectangle`
- Silk4J does not support testing native mobile applications with multiple web views.
- When testing on iOS, the state of the `isVisible` property is always true, even if the element is not visible.
- When testing on iOS, a swipe action with multiple steps swipes to a point, releases the mouse pointer and then swipes to the next point. On prior versions of iOS, the action does not release the mouse pointer between the swipes.
- When testing on iOS, Silk4J does not support any multi-touch actions except pinch.
- When testing on iOS, Silk4J does not support the `pinchIn` method.
- When testing on iOS, you can only accept or dismiss alert dialog boxes. If no **Cancel** button is available and Silk4J cannot dismiss the dialog, the default action is to accept the dialog.
- When testing on Android, Silk4J does not provide automated synchronization for controls of the [Animation](#) class.
- When testing toasts on Android, the following limitations apply:
 - During recording, Silk4J always displays the rectangle for the toast in the lowest quarter of the **Recording** window, independent of the actual position of the toast.
 - During recording and replay, the detection of a toast by Silk4J always has a duration of five seconds, even if the toast appears in a shorter time period.
- When testing on iOS, Silk4J does not provide automated synchronization for controls that call the `UIView.animate` function or the `UIView.animateWithDuration` function.

You can workaround this issue by increasing the speed of the animation in the app delegate:

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
    //...
    if NSProcessInfo.processInfo().environment["automationName"] == "Silk
Test" {
        // Speed animations up (recommended)
        window!.layer.speed = 100;
    }
}
```

Micro Focus does not recommend disabling such animations completely, as this might change the applications behavior. However, if speeding up the animation does not resolve the synchronization issue, you could completely disable animations in the app delegate as follows:

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
    //...
    if NSProcessInfo.processInfo().environment["automationName"] == "Silk
Test" {
        UIView.setAnimationsEnabled(false)
    }
}
```

- When testing on iOS, the following additional limitations apply:
 - You might experience performance decreases while recording and replaying tests.
 - Due to internal changes in iOS, the locators of some controls might have changed, and some of your existing tests might break.
 - Text fields that are not in focus might not be recognized as text fields. To ensure that text fields are recognized correctly, set the focus on the text fields, for example by clicking on a text field before trying to interact with it.

Dynamically Invoking Methods for Native Mobile Apps

Dynamic invoke enables you to directly call methods of the underlying Appium WebDriver for a mobile native app. This is useful whenever an Appium WebDriver method is not exposed through the Silk4J API.

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.

Supported Methods

- When testing a native mobile application on Android, Silk4J supports the methods available in the `AndroidDriver` class of the [Appium Java-client API](#).
- When testing a native mobile application on iOS, Silk4J supports the methods available in the `IOSDriver` class of the [Appium Java-client API](#).

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 `ScreenOrientation`, you can use the string values `LANDSCAPE` or `PORTRAIT`.

- 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.

Returned Values

The following values are returned for 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.

Example

The following code sample contains some common examples for using dynamic invoke.

```
// Java code
MobileDevice device = desktop.find("//MobileDevice");

// Getting the page source
String pageSource = (String) device.invoke("getPageSource");

// Resetting an app
device.invoke("resetApp");

// Changing the device orientation
device.invoke("rotate", "LANDSCAPE");
device.invoke("rotate", "PORTRAIT");

// Dynamic invoke on MobileObject (calls get redirected to the
underlying web element for WebDriver)
device.<MobileObject> find("//MobileObject[@caption='CheckBox
2']").invoke("click");
```

Clicking on Objects in a Mobile Website

When clicking on an object during the recording and replay of an automated test, a mobile website presents the following challenges in comparison to a desktop website:

- Varying zoom factors and device pixel ratios.
- Varying screen sizes for different mobile devices.
- Varying font and graphic sizes between mobile devices, usually smaller in comparison to a website in a desktop browser.
- Varying pixel size and resolution for different mobile devices.

Silk4J enables you to surpass these challenges and to click the appropriate object on a mobile website.

When recording a test on a mobile device, Silk4J does not record coordinates when recording a `Click`. However, for cross-browser testing, coordinates are allowed during replay. You can also manually add coordinates to a `Click`. Silk4J interprets these coordinates as the HTML coordinates of the object. To click on the appropriate object inside the `BrowserWindow`, during the replay of a test on a mobile device, Silk4J applies the current zoom factor to the HTML coordinates of the object. The device pixel coordinates are the HTML coordinates of the object, multiplied with the current zoom factor.

If the object is not visible in the currently displayed section of the mobile website, Silk4J scrolls to the appropriate location in the website.

Example

The following code shows how you can test a `DomButton` with a fixed size of 100 x 20 px in your HTML page.

```
DomButton domButton = desktop.find("locator for the button");
domButton.click(MouseButton.LEFT, new Point(50, 10));
```

During replay on a different mobile device or with a different zoom factor, the `DomButton` might for example have an actual width of 10px on the device screen. Silk4J clicks in the middle of the element when using the code above, independent of the current zoom factor, because Silk4J interprets the coordinates as HTML coordinates and applies the current zoom factor.

Using Existing Mobile Web Tests

Silk Test 17.0 or later uses a different approach to mobile web testing than previous versions of Silk Test. This change might result in your old mobile web tests no longer working on Silk Test 17.0 or later. This topic describes some of the changes that were introduced with Silk Test 17.0 and provides guidance on changing existing mobile web tests with Silk Test 17.0 or later.

The following changes for mobile web testing were introduced with Silk Test 17.0:

- With previous versions of Silk Test, you were able to test on iOS devices that were connected by USB to a Windows machine. With Silk Test 17.0 or later, you can only test on iOS devices that are connected to an OSX machine (Mac).
- If you have tested mobile web applications on an Android device with a previous version of Silk Test, you have to manually remove the proxy from the Android device to test a web application with Silk Test 17.0 or later. Silk Test 17.0 or later no longer requires a proxy, and if the proxy is set, the message `Unable to connect to the proxy server` displays on the device.

.NET Support

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

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Windows Forms Support

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Object Recognition

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

Supported Controls

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

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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Custom Attributes for Windows Forms Applications

Windows Forms applications use the predefined automation property `automationId` to specify a stable identifier for the Windows forms control.

Silk4J automatically will use this property for identification in the locator. Windows Forms application locators look like the following:

```
/FormsWindow//PushButton[@automationId='btnBasicControls']
```

Dynamically Invoking Windows Forms Methods

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

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

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

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

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

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

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

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

The invoke Method

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

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

First Example for the invoke Method

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

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

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

Second Example for the invoke Method

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

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

Third Example for the invoke Method

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

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

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

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

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

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

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

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

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

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

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

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

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

The `invokeMethods` Method

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

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

Example: Getting the Text Contents of a Cell in a Custom Data Grid

To get the text contents of a cell of a custom data grid from the Infragistics library, you can use the following C# code in the AUT:

```
string cellText =  
dataGrid.Rows[rowIndex].Cells[columnIndex].Text;
```

The following C# code sample gets the text contents of the third cell in the first row:

```
string cellText = dataGrid.Rows[0].Cells[2];
```

Scripting the same example by using the `invokeMethods` method generates a relatively complex script, because you have to pass five methods with their corresponding parameters to the `invokeMethods` method:

```
WPFControl dataGrid = mainWindow.find("//  
WPFControl[@automationId='Custom Data Grid']");  
  
// Get text contents of third cell in first row.  
int rowIndex = 0;  
int columnIndex = 2;  
  
List<String> methodNames = Arrays.asList("Rows", "get_Item",  
"Cells", "get_Item", "Text");  
List<List<Object>> parameters = Arrays.asList(new  
ArrayList<Object>(), Arrays.<Object>asList(rowIndex), new  
ArrayList<Object>(), Arrays.<Object>asList(rowIndex), new  
ArrayList<Object>());  
  
String cellText = (String) dataGrid.invokeMethods(methodNames,  
parameters);
```

A better approach in such a case is to add code to the application under test and then to use the `invokeMethods` method. For this example, add the `getCellText` method to the AUT:

```
// C# code, if the AUT is implemented in C#.  
public static string  
GetCellText(Infragistics.Win.UltraWinGrid.UltraGrid dataGrid,  
int rowIndex, int columnIndex) {  
    return dataGrid.Rows[rowIndex].Cells[columnIndex].Text;  
}  
  
' VB code, if the AUT is implemented in VB.  
public static string  
GetCellText(Infragistics.Win.UltraWinGrid.UltraGrid dataGrid,
```

```
int rowIndex, int columnIndex) {  
    return dataGrid.Rows[rowIndex].Cells[columnIndex].Text;  
}
```

To get the text contents of the cell, dynamically invoke the `GetCellText` method from your test script:

```
String cellText = (String) mainWindow.invoke("GetCellText",  
dataGrid, rowIndex, columnIndex);
```

For additional information, see *Adding Code to the Application Under Test to Test Custom Controls*.

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.Visibility` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.

- .NET structs and objects

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

- Other controls

Control parameters can be passed or returned as `TestObject`.

Returned Values

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

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

Windows Presentation Foundation (WPF) Support

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported Controls

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

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

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

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

For additional information about WPF, refer to [MSDN](#).

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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards `?` and `*`.

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

Object Recognition

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

```
WPFButton[@automationId='okButton']"
```

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

Attribute Type	Description	Example
<code>automationId</code>	An ID that was provided by the developer of the test application.	<code>//WPFButton[@automationId='okButton']"</code>
<code>name</code>	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.	<code>//WPFButton[@name='okButton']"</code>

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

Custom Attributes for WPF Applications

WPF applications use the predefined automation property `AutomationProperties.AutomationId` to specify a stable identifier for the WPF control as follows:

```
<Window x:Class="Test.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
<Grid>
<Button AutomationProperties.AutomationId="AID_buttonA">The
Button</Button>
</Grid>
</Window>
```

Silk4J automatically uses this property for identification in the locator. WPF application locators look like the following:

```
//WPFWindow[@caption='MainWindow']//WPFButton[@automationId='AID_buttonA']
```

Classes that Derive from the WPFItemsControl Class

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

- Working with the control

Most controls contain methods and properties for typical use cases. The items are identified by text or index.

- Working with individual items, such as `WPFListBoxItem`, `WPFTreeViewItem`, or `WPFMenuItem`

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

Custom WPF Controls

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

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

- Deriving classes from `UserControl`

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

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

Silk4J treats these controls as an instance of the standard WPF control that they derive from. Record, playback, and recognition of children may not work if the user control behavior differs significantly from its base class implementation.

- Using standard controls that use templates to change their visual appearance

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

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

Dynamically Invoking WPF Methods

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

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

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

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

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

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



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



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

The invoke Method

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

- Public methods that the MSDN defines for the control.
- Public static methods that the MSDN defines.

- User-defined public static methods of any type.

First Example for the invoke Method

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

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

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

Second Example for the invoke Method

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

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

Third Example for the invoke Method

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

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

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

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

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

- The application developer can compile the code for the class into the AUT. The assembly needs to be already loaded.

- You can create a new assembly that is loaded into the AUT during test execution. To load the assembly, you can use the following code:

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

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

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

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

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

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

The invokeMethods Method

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

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

Example: Getting the Text Contents of a Cell in a Custom Data Grid

To get the text contents of a cell of a custom data grid from the Infragistics library, you can use the following C# code in the AUT:

```
string cellText =
dataGrid.Rows[rowIndex].Cells[columnIndex].Text;
```

The following C# code sample gets the text contents of the third cell in the first row:

```
string cellText = dataGrid.Rows[0].Cells[2];
```

Scripting the same example by using the invokeMethods method generates a relatively complex script, because you have to pass five methods with their corresponding parameters to the invokeMethods method:

```
WPFControl dataGrid = mainWindow.find("//
WPFControl[@automationId='Custom Data Grid']");

// Get text contents of third cell in first row.
int rowIndex = 0;
int columnIndex = 2;

List<String> methodNames = Arrays.asList("Rows", "get_Item",
"Cells", "get_Item", "Text");
List<List<Object>> parameters = Arrays.asList(new
ArrayList<Object>(), Arrays.<Object>asList(rowIndex), new
ArrayList<Object>(), Arrays.<Object>asList(rowIndex), new
ArrayList<Object>());

String cellText = (String) dataGrid.invokeMethods(methodNames,
parameters);
```

A better approach in such a case is to add code to the application under test and then to use the invokeMethods method. For this example, add the getCellText method to the AUT:

```
// C# code, if the AUT is implemented in C#.
public static string
```

```
GetCellText(Infragistics.Win.UltraWinGrid.UltraGrid dataGrid,
int rowIndex, int columnIndex) {
    return dataGrid.Rows[rowIndex].Cells[columnIndex].Text;
```

' VB code, if the AUT is implemented in VB.

```
public static string
GetCellText(Infragistics.Win.UltraWinGrid.UltraGrid dataGrid,
int rowIndex, int columnIndex) {
    return dataGrid.Rows[rowIndex].Cells[columnIndex].Text;
```

To get the text contents of the cell, dynamically invoke the `GetCellText` method from your test script:

```
String cellText = (String) mainWindow.invoke("GetCellText",
dataGrid, rowIndex, columnIndex);
```

For additional information, see *Adding Code to the Application Under Test to Test Custom Controls*.

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.Visibility` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.

- .NET structs and objects

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

- WPF controls

WPF control parameters can be passed as `TestObject`.

Returned Values

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

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

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

Example

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

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

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

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

Setting WPF Classes to Expose During Recording and Playback

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

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

1. Click **Silk4J > Edit Options**.
2. Click the **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 Pre-Fill During Recording and Replaying

Defines whether items in a *WPFItemsControl*, like *WPFComboBox* or *WPFListBox*, are pre-filled during recording and playback. WPF itself lazily loads items for certain controls, so these items are not available for Silk4J if they are not scrolled into view. Turn pre-filling on, which is the default setting, to additionally access items that are not accessible without scrolling them into view. However, some applications have problems when the items are pre-filled by Silk4J in the background, and these applications can therefore crash. In this case turn pre-filling off.

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

2. Click the **WPF** tab.
3. In the **Pre-fill items** area, check the **OPT_WPF_PREFILL_ITEMS** check box.
4. Click **OK**.

Silverlight Application Support

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

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

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

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

For information about new features, supported platforms, and tested versions, 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*.



Note: With Silk Test 14.0 or later, Silk4J recognizes only Silverlight controls that are available for interaction and visible on the screen. This change might change the behavior of tests that were recorded with a Silk Test version prior to Silk Test 14.0. To run such tests with Silk Test 14.0 or later, remove all invisible or not yet available Silverlight controls from the tests.

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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

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

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

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

Attribute Type	Description	Example
automationId	An identifier that is provided by the developer of the application under test. The Visual Studio designer automatically assigns an <i>automationId</i> 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 <i>automationId</i> or <i>name</i> attribute instead of the <i>caption</i> .	//SLButton[@caption="Ok"]
className	The simple .NET class name (without namespace) of the Silverlight control. Using the <i>className</i> 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
<code><Button>Ok</Button></code>	<code>//SLButton[@caption="Ok"]</code>
<code><Button Name="okButton">Ok</Button></code>	<code>//SLButton[@automationId="okButton"]</code>
<code><Button AutomationProperties.AutomationId="okButton">Ok</Button></code>	<code>//SLButton[@automationId="okButton"]</code>
<code><Button AutomationProperties.Name="okButton">Ok</Button></code>	<code>//SLButton[@name="okButton"]</code>

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.Visibility` 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("SelectedItemPattern.Select");  
mySilverlightObject.GetProperty("IsPassword");
```

Scrolling in Silverlight

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

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

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

Controls that support scrolling

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

Examples

The following command scrolls a list box to the bottom:

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

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

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

Controls that do not support scrolling

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

Examples

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

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

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

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

Troubleshooting when Testing Silverlight Applications

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

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

Reason	Solution
You use a 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.	<p>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 <code>windowless</code> parameter in the object tag of the HTML or ASPX file, in which the Silverlight application is hosted, to <code>false</code>.</p> <p>The following sample code sets the <code>windowless</code> parameter to <code>false</code>:</p> <pre><object ...> <param name="windowless" value="false"/> ... </object></pre>

Visual COBOL Support

Silk4J supports recording and replaying tests against Visual COBOL applications. You can also use the Silk4J APIs from .NET COBOL to script automated tests for the following Visual COBOL applications:

- Dialog system applications.
- CGI applications.
- Cobol Win32 applications.
- .NET COBOL - WPF and Windows Forms applications.
- COBOL JVM - Swing applications.
- Non-COBOL front-end applications calling back-end COBOL.



Note: For some controls, Silk4J provides only low-level recording support.

For information on the supported versions of Visual COBOL, refer to the [Release Notes](#).

Rumba Support

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

When testing with Rumba, please consider the following:

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

For a complete list of the record and replay controls available for Rumba testing, see the [Rumba Class Reference](#).

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Enabling and Disabling Rumba

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

Enabling Support

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

1. Install Rumba desktop client software version 8.1 or later.
2. Click (in Microsoft Windows 7) **Start > Programs > Silk > Silk Test > Administration > Rumba plugin > Enable Silk Test Rumba plugin** or (in Microsoft Windows 10) **Start > Silk > Enable Silk Test Rumba plugin**.

Disabling Support

Click (in Microsoft Windows 7) **Start > Programs > Silk > Silk Test > Administration > Rumba plugin > Disable Silk Test Rumba plugin** or (in Microsoft Windows 10) **Start > Silk > Disable Silk Test Rumba plugin**.

Locator Attributes for Identifying Rumba Controls

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

caption	The text that the control displays.
priorlabel	Since input fields on a form normally have a label explaining the purpose of the input, the intention of priorlabel is to identify the text input field, RumbaTextField , by the text of its adjacent label field, RumbaLabel . If no preceding label is found in the same line of the text field, or if the label at the right side is closer to the text field than the left one, a label on the right side of the text field is used.
StartRow	This attribute is not recorded, but you can manually add it to the locator. Use StartRow to identify the text input field, RumbaTextField , that starts at this row.
StartColumn	This attribute is not recorded, but you can manually add it to the locator. Use StartColumn to identify the text input field, RumbaTextField , that starts at this column.
All dynamic locator attributes.	For additional information on dynamic locator attributes, see <i>Dynamic Locator Attributes</i> .



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

Testing a Unix Display

Unix displays in Rumba are completely text-based, and provide no UI controls except the main **RUMBA screen** control. To replay a test on a Unix display, you can use the `sendKeys` method to send keys to the Unix display. Silk4J does not support recording on a Unix display.

SAP Support

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

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

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported Controls

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

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

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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Dynamically Invoking SAP Methods

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

Supported Parameter Types

The following parameter types are supported:

- All built-in Silk4J types

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

- UI controls

UI controls can be passed or returned as `TestObject`.

Returned Values

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

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

Dynamically Invoking Methods on SAP Controls

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

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

For additional information on the **SAP GUI Scripting** tool, refer to the SAP documentation.

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

Examples

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

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

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

```
.SapToolBarControl("shell  
ToolBarControl").invoke("pressContextButton", "TEST")  
.SapToolBarControl("shell  
ToolBarControl").invoke("selectContextMenuItem", "subsub2")
```

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

Configuring Automation Security Settings for SAP

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

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

Universal Windows Platform Support

Describes the built-in support for testing Microsoft Windows 10 apps that are using the Universal Windows Platform API.

UWP is an API created by Microsoft that allows developing apps that can be purchased and downloaded through the Microsoft Store.

Silk4J provides built-in support for testing UWP apps on the following operating systems:

- Microsoft Windows 10
- Microsoft Windows Server 2019



Note: Silk Test does not support testing Universal Windows Platform (UWP) apps (also known as Windows Store apps or Metro-style apps) on Microsoft Windows 8 and Microsoft Windows 8.1.

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported controls

Silk4J includes record and replay support for UWP controls that are based on UIA controls, as well as for the following controls:

- `UWPComboBoxItem`
- `UWPComboBox`

Application configuration

The application configuration for a UWP app includes the following:

Executable The executable for a UWP app is a combination of the name of the app, the publisher ID, and a part of the app identifier that uniquely identifies the app in the context of the package. For example, a UWP app with the name *Microsoft.WindowsCalculator*, the publisher ID *8wekyb3d8bbwe*, and the app identifier *App* has the executable *Microsoft.WindowsCalculator_8wekyb3d8bbwe!App*.

Executable pattern The executable pattern for a UWP app is similar to the executable. For example, a UWP app could have the executable pattern *Microsoft.WindowsCalculator_8wekyb3d8bbwe!App* or **Microsoft.WindowsCalculator_8wekyb3d8bbwe!App*.

An individual window of a UWP app can often be another UWP app. To test such a window, you need to configure an additional application configuration. If Silk4J does not recognize a window in a UWP app,

open the **Select Application** dialog box while the window is open and check whether the **Windows** tab includes an entry for the window that is not recognized. Then add an application configuration for the window.

Troubleshooting when Testing UWP Apps

Why does Silk4J not recognize a window in a UWP app?

An individual window of a UWP app can often be another UWP app. To test such a window, you need to configure an additional application configuration. If Silk4J does not recognize a window in a UWP app, open the **Select Application** dialog box while the window is open and check whether the **Windows** tab includes an entry for the window that is not recognized. Then add an application configuration for the window.

Windows API-Based Application Support

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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Supported Controls

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

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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

Determining the priorLabel in the Win32 Technology Domain

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

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

Testing Embedded Chrome Applications

An embedded Chrome application is a desktop application with an embedded web browser engine that is based on the Chromium core. Such applications enable you to add web browser capabilities to a desktop application. You can create such an app by using for example the Chromium Embedded Framework (CEF) or the Electron framework.

Silk4J provides full support for testing embedded Chrome applications that allow remote debugging through the `--remote-debugging-port` command line argument. Silk4J does not support testing embedded Chrome applications that are based on Java, for example Java AWT and Swing applications.

To test an embedded Chrome application with Silk4J, you have to set the debugging ports for the executable of the application. Start the application from the command line and set the remote debugging port.

- Silk4J checks if the `-remote-debugging-port` argument is set in the command line arguments of the embedded Chrome application. If the argument is set, Silk4J automatically sets the **Enable embedded Chrome support** field to the appropriate executable and debugging port.
- If the `-remote-debugging-port` argument is not set in the command line arguments of the embedded Chrome application, you have to manually specify the executable and the port in the **Enable embedded Chrome support** field:
 1. In the Silk4J UI, select **Edit Options**.
 2. In the **Options** dialog, select the **Advanced** tab.
 3. In the **Enable embedded Chrome support** option, specify the executable and the port as a comma-separated value pair:

```
<application name>.exe=<port number>
```



Note: You cannot test embedded Chrome applications that do not allow remote debugging with Silk4J.



Note: Silk4J does not support testing non-browser menus of Electron apps.

Example

For example, you can start the application *myApp* from the command line as follows:

```
myApp.exe --remote-debugging-port=9222
```

You can then specify the executable and port in the **Enable embedded Chrome support** option as follows:

```
myApp.exe=9222
```

Microsoft Foundation Class Support

The class ID of a Microsoft Foundation Class (MFC) control might change over time and therefore cannot be used to generate a stable locator. To avoid generating unstable locators, Silk4J uses the following attributes for the locators:

- The MFC class name, if the Windows class name of the MFC control starts with `Afx:`.
- The Windows class name, if the Windows class name of the MFC control does not start with `Afx:`.

Silk4J only supports MFC version 140, and only supports the following combinations:

- Release, x86, MBCS
- Release, x86, Unicode
- Debug, x86, MBCS
- Debug, x86, Unicode
- Release, x64, MBCS
- Release, x64, Unicode
- Debug, x64, MBCS
- Debug, x64, Unicode



Note: To execute existing tests with MFC control locators that have been generated with Silk4J 18.5 or prior, set the `OPT_COMPATIBILITY` option in the affected test scripts to version 18.5.0 or prior:

```
'VB .NET code
Agent.SetOption( "OPT_COMPATIBILITY", "18.5.0" )
```

Cross-Browser Testing

With Silk4J, you can easily verify the functionality of even the most advanced web application across a variety of browsers, with a single, portable test script. Silk4J provides leading support for effective and maintainable cross-browser testing with modern web technologies.

One of the main challenges in test automation is to create and maintain test cost effectively. As different browsers behave differently, web application validation is hard to carry out productively. Silk4J enables you to focus on writing tests, as it handles the following three areas of cross-browser testing:

Built-in synchronization

This enables you to create scripts that run on all supported browsers, without the need to manually synchronize against asynchronous events, which are typical of highly dynamic web applications such as AJAX, or HTML5. Silk4J supports synchronization modes for HTML or AJAX as well as all major web environments including Apache Flex, Microsoft Silverlight, and HTML5/AJAX. For additional information, see [Page Synchronization for xBrowser](#).

Unified object model

Silk4J enables you to create and maintain a test which runs across a wide range of different browsers. A unified object model across all browsers gives you the ability to focus on a single browser when you create or maintain a test. Silk4J ensures that the object you interact with is accessible in the same way on all the other browsers, which saves time and enables you to focus on testing rather than on finding workarounds for different browsers.

Out-of-the-box recording of a cross-browser script

Record a script once and replay it in all the other browsers, without any modifications. This significantly reduces the time and effort it takes to create and maintain test scripts. Nothing is simulated - testing is carried out across real browsers, which ensures that the test behaves exactly as it does for your end user.

With Silk4J, you can replay tests against web applications that use:

- Internet Explorer.
- Mozilla Firefox, both on Microsoft Windows and on macOS.
- Google Chrome, both on Microsoft Windows and on macOS.
- Microsoft Edge.
- Chrome for Android on an Android device.
- Apple Safari, both on macOS and on an iOS device.
- Embedded browser controls.



Note: You can record tests for web applications using one of the following browsers:

- Internet Explorer.
- Microsoft Edge.
- Mozilla Firefox, both on Microsoft Windows and on macOS.
- Google Chrome, both on Microsoft Windows and on macOS.
- A mobile browser on a mobile device.

When recording a script for cross-browser testing, Micro Focus recommends using Google Chrome, Mozilla Firefox, or Microsoft Edge, as a script recorded with Silk4J against Internet Explorer might slightly differ in comparison to a script recorded on one of the other browsers.



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

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Sample Applications

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

- <http://demo.borland.com/InsuranceWebExtJS/>
- <http://demo.borland.com/gmopost>
- <http://demo.borland.com/gmoajax>

Selecting the Browser for Test Replay

You can define the browser that is used for the replay of a test in the following ways:

- If you execute a test from the UI of Silk4J and the **Select Browser** dialog box displays, the browser selected in the dialog box is used, and Silk4J ignores which browser is set in the test script.
- If the **Select Browser** dialog box is disabled, because the **Don't show again** is checked, the application configurations in the individual test scripts determine the browser that is used to execute the tests.



Note: To re-enable the **Select Browser** dialog box, click **Silk4J > Edit Application Configurations** and check the **Show 'Select Browser' dialog before record and playback** check box

- If you execute a script from the command line or from a Continuous Integration (CI) server, specify the connection string in the application configuration of the script.

To overwrite the browser that is specified in the application configuration, use the `silktest.configurationName` environment variable.

- If you execute a test from Silk Central, create a configuration suite with a configuration for each browser that you want to test. Then specify the appropriate configuration name. For additional information, refer to the [Silk Central Help](#).

Examples of setting the browser by using the `silktest.configurationName` environment variable

- To use Internet Explorer as the browser, type:

```
SET silktest.configurationName=InternetExplorer
```



Note: Internet Explorer

- To use Microsoft Edge as the browser, type:

```
SET silktest.configurationName=Edge
```

- To use Mozilla Firefox as the browser, type:

```
SET silktest.configurationName=Firefox
```

- To use Google Chrome as the browser, type:

```
SET silktest.configurationName=GoogleChrome
```

- To use Apple Safari on a Mac as the browser, type:

```
SET silktest.configurationName=host=10.0.0.1 - Safari
```

In this example, the `host` is the Mac, on which you want to test Apple Safari. The host needs to be connected as a remote location to the machine on which Silk4J is running. For additional information, see *Editing Remote Locations*.

- To use Google Chrome on an Android device as the browser, use a connection string. For example, if the device ID is 11111111 and the device is connected to the remote machine with the IP address 10.0.0.1, type:

```
SET  
silktest.configurationName="platformName=Android;deviceName=Mo  
toG3;deviceId=11111111;host=10.0.0.1 - Chrome"
```

- To use the Original Android Stock (AOSP) Browser on an Android device, use a connection string. For example, if the device ID is 11111111 and the device is connected to the remote machine with the IP address 10.0.0.1, type:

```
SET  
silktest.configurationName="platformName=Android;deviceName=Mo  
toG3;deviceId=11111111;host=10.0.0.1 - AndroidBrowser"
```

- To use Apple Safari on an iOS device as the browser, use a connection string. For example, if the device ID is 11111111 and the device is connected to the remote machine with the IP address 10.0.0.1, type:

```
SET  
silktest.configurationName="platformName=iOS;deviceName=iPad  
mini;deviceId=11111111;host=10.0.0.1"
```

Additionally, you have to specify the browser in the application configuration.



Tip: For all examples, you can also set the browser by setting the Java System property `-Dsilktest.configurationName` instead of setting the environment variable `silktest.configurationName`. For example, to use Apple Safari on a Mac as the browser, you can also type:

```
-Dsilktest.configurationName=host=10.0.0.1 -  
Safari
```

To execute the tests from the command line, type:

```
java -cp "...\junit.jar;...  
\org.hamcrest.core_1.3.0.v201303031735.jar;C:  
\Program Files (x86)\Silk\SilkTest\ng\JTF  
\silktest-jtf-nodeps.jar;...\mytests\bin" -  
Dsilktest.configurationName="host=10.0.0.1 -  
Safari" org.junit.runner.JUnitCore Tests
```



Tip: Open the **Select Browser** dialog box, for example by starting to replay or record from the Silk4J UI, to see a list of the browsers that are currently available on your system.

Test Objects for xBrowser

Silk4J uses the following classes to model a web application:

Class	Description
BrowserApplication	Exposes the main window of a web browser and provides methods for tabbing.
BrowserObject	Represents the base class for all objects that are contained within a BrowserApplication.
BrowserWindow	Provides access to tabs and embedded browser controls and provides methods for navigating to different pages.
DomElement	Exposes the DOM tree of a web application, including frames, and provides access to all DOM attributes. Specialized classes are available for several DOM elements.
DomButton	Represents the top-level container for a web page. It exposes the DOM tree through DomElement.
DomCheckBox	Represents all DOM elements that were specified using the <code><input type='checkbox'></code> tag.
DomForm	Represents all DOM elements that are specified using the <code><form></code> tag.
DomLink	Represents all DOM elements that were specified using the <code><a></code> tag.
DomListBox	Represents all DOM elements that were specified using the <code><select></code> tag.
DomRadioButton	Represents all DOM elements that were specified using the <code><input type='radio'></code> tag.
DomTable	Represents all DOM elements that were specified using the <code><table></code> tag.
DomTableRow	Represents all DOM elements that were specified using the <code><tr></code> tag.
DomTextField	Represents all DOM elements that were specified using one of the following tags: <ul style="list-style-type: none">• <code><input type='text'></code>• <code><input type='password'></code>• <code><input type='file'></code>• <code><textarea></code>

Object Recognition for xBrowser Objects

The xBrowser technology domain supports dynamic object recognition.

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

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

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

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

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

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

Page Synchronization for xBrowser

Synchronization is performed automatically 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`
- `WaitForPropertyNotEquals`
- `WaitForDisappearance`
- `WaitForChildDisappearance`
- `WaitForScreenshotStable`

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

Troubleshooting

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

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

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"));
```

- `OPT_XBROWSER_SYNC_MODE`
- `OPT_XBROWSER_SYNC_EXCLUDE_URLS`
- `OPT_SYNC_TIMEOUT`

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	<code>DomClick</code>	<code>Click</code>
	<code>DomDoubleClick</code>	<code>DoubleClick</code>
	<code>DomMouseMove</code>	<code>MoveMouse</code>
		<code>PressMouse</code>
		<code>ReleaseMouse</code>
Keyboard Actions	not available	<code>TypeKeys</code>
Specialized Functions	<code>Select</code>	not available
	<code>SetText</code>	
	etc.	

Setting Mouse Move Preferences

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

1. Click **Silk4J > Edit Options**.
2. Click the plus sign (+) next to **Record** in the **Options** menu tree. The **Record** options display in the right side panel.
3. Click **Recording**.
4. To record mouse move actions, check the `OPT_RECORD_MOUSEMOVES` option.
Silk4J will only record mouse move events that cause changes to the hovered element or its parent in order to keep scripts short.
5. If you record mouse move actions, in the **Record mouse move delay** text box, specify how many milliseconds the mouse has to be motionless before a `MoveMouse` action is recorded
By default this value is set to 200.
Mouse move actions are only recorded if the mouse stands still for this time. A shorter delay will result in more unexpected mouse move actions, a longer delay will require you to keep the mouse still to record an action.
6. Click **OK**.

Browser Configuration Settings for xBrowser

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

Increase replay speed

Use `about:blank` as home page instead of a slowly loading Web page.

Avoid unexpected behavior of the browser

- Disable pop up windows and warning dialog boxes.
- Disable auto-complete features.

- Disable password wizards.

Prevent malfunction of the browser

Disable unnecessary third-party plugins.

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

Internet Explorer

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

Tab	Option	Configuration	Comments
General	Home page	Set to <code>about:blank</code> .	Minimize start up time of new tabs.
General	Tabs	<ul style="list-style-type: none"> • Disable warning when closing multiple tabs. • Enable to switch to new tabs when they are created. 	<ul style="list-style-type: none"> • Avoid unexpected dialog boxes. • Links that open new tabs might not replay correctly otherwise.
Privacy	Pop-up blocker	Disable pop up blocker.	Make sure your Web site can open new windows.
Content	AutoComplete	Turn off completely	<ul style="list-style-type: none"> • Avoid unexpected dialog boxes. • Avoid unexpected behavior when typing keys.
Programs	Manage add-ons	Only enable add-ons that are absolutely required.	<ul style="list-style-type: none"> • Third-party add-ons might contain bugs. • Possibly not compatible to Silk4J.
Advanced	Settings	<ul style="list-style-type: none"> • Disable Automatically check for Internet Explorer updates. • Enable Disable script debugging (Internet Explorer). • Enable Disable script debugging (Other). • Disable Enable automatic crash recovery. • Disable Display notification about every script error. • Disable all Warn ... settings 	Avoid unexpected dialog boxes.



Note: Recording a Web application in Internet Explorer with a zoom level different to 100% might not work as expected. Before recording actions against a Web application in Internet Explorer, set the zoom level to 100%.

Mozilla Firefox

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



Note: To avoid unexpected behavior when testing web applications, disable auto updates for Mozilla Firefox. For additional information, see [Stop automatic updates](#).

Google Chrome

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



Note: To avoid unexpected behavior when testing web applications, disable auto updates for Google Chrome. For additional information, see [Turning Off Auto Updates in Google Chrome](#).

Configuring the Locator Generator for xBrowser

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

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

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

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

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

3. Click the **Browser** tab.
4. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording.

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

Separate attribute names with a comma.

5. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording.

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

Separate attribute values with a comma.

6. Click **OK**.

You can now record or manually create a test case.

Connection String for a Remote Desktop Browser

The *connection string* specifies the remote desktop browser that is used for testing. When testing a web application in a remote browser, Silk4J uses the connection string to connect to the remote location. The connection string is typically part of the application configuration. You can set the connection string when you configure the web application that you want to test. To change the connection string, you can use the **Edit Application Configuration** dialog box.

When testing a web application in a remote browser, the connection string includes only the host, which means the IP address or the host name of the remote machine, for example 10.0.0.1. To select the correct browser, Silk4J uses the connection string in combination with the browser type, which you can also specify in the **Edit Application Configuration** dialog box.

The host name is case-insensitive.



Note: Remote desktop browser testing is only supported for Microsoft Edge on a remote Microsoft Windows machine, and for Apple Safari on a remote Mac.

Connection string example

```
"host=10.0.0.1"
```

Testing Browsers on a Remote Windows Machine

To test Microsoft Edge, Google Chrome, or Mozilla Firefox on a remote Windows machine, Silk4J needs to be installed on the remote machine. Microsoft Edge does only run on a Microsoft Windows 10 machine. Silk4J supports testing Mozilla Firefox 55 or prior on a remote Windows machine.

1. On the local machine, from which you want to test the browser, add the remote Windows machine as a remote location.

For additional information, see [Editing Remote Locations](#).

2. If the Silk Test information service is already running with administrator privileges on the remote machine, disable the Silk Test information service.

- a) Sign in to the remote machine as an administrator.
- b) Open the **Control Panel** (icons view).
- c) Click **Administrative Tools**.
- d) Double-click **Services**.
- e) Double click on the Silk Test information service.
- f) If the service shows a status of running, click **Stop** and wait until the service status shows as stopped.
- g) Change the **Startup type** to **Disabled**.
- h) Click **OK**.

3. Start the Silk Test information service with medium integrity level, which means without administrator privileges.

- a) Open a file explorer and navigate to %OPEN_AGENT_HOME%/InfoService.
For example, C:\Program Files (x86)\Silk\SilkTest\ng\InfoService.
- b) Double-click `infoservice_start.bat`.

4. You can now select the browser on the remote machine in the **Select Application** dialog box.



Note: Ensure that the Silk Test information service is running in a user session, and not in the services session. The Silk Test information service requires UI interaction to be enabled. The services session, which is session 0, does not enable UI interaction.

Testing Google Chrome or Mozilla Firefox on a Mac

To test Google Chrome or Mozilla Firefox on a remote macOS machine, the Silk Test information service needs to be installed on the remote machine. For additional information, see [Installing the Silk Test Information Service on a Mac](#).

1. On the local machine, from which you want to test Google Chrome or Mozilla Firefox, add the remote macOS machine as a remote location.

For additional information, see [Editing Remote Locations](#).

2. You can now select Google Chrome or Mozilla Firefox on the remote macOS machine in the **Select Application** dialog box.



Note: Ensure that the Silk Test information service is running in a user session, and not in the services session. The Silk Test information service requires UI interaction to be enabled. The services session, which is session 0, does not enable UI interaction.

For information on the prerequisites and limitations when testing on Google Chrome, refer to the topics in the section [Testing with Google Chrome](#). For information on the prerequisites and limitations when testing on Mozilla Firefox, refer to the topics in the section [Testing with Mozilla Firefox](#).

Setting Capabilities for WebDriver-Based Browsers

If you are testing a web application on a WebDriver-based browser, you can customize and configure the browser session by setting the capabilities.

In Silk4J, you can specify WebDriver capabilities in the connection string for the following browser types:

- Google Chrome
- Mozilla Firefox

For information on the available options and capabilities for Mozilla Firefox 48 or later, see <https://github.com/mozilla/geckodriver>. For information on the available options and capabilities for Google Chrome, see [Capabilities & ChromeOptions](#).

To set the capabilities in Silk4J:

1. Select the project which corresponds to the web application for which you want to change the capabilities.
2. Edit the connection string in the base state of the project.

You can edit the connection string in the following ways:

- By using the **Edit Application Configurations** dialog, for example if you want to record actions against a customized browser.
- In a script, if you only want to execute the tests in the script against the customized browser.

For additional information, see [Base State](#).

3. Execute the script to start the browser with the specified options and capabilities.

Examples

You can add the following code to the base state in a script to automatically download executables from Mozilla Firefox:

```
baseState.setConnectionString(
    "moz:firefoxOptions="
    + "{"
    + "  \"prefs\": {"
    + "    \"browser.download.folderList\": 2,"
    + "    \"browser.helperApps.neverAsk.saveToDisk\":
    + \"application/octet-stream\"
    + "  }"
    + "};");
```

You can add the following code to the base state in a script to specify the download folder for Mozilla Firefox:

```
baseState.setConnectionString("moz:firefoxOptions={\"prefs\":
{ \"browser.download.dir\" : \"C:/Download\" } };");
```

You can add the following code to the base state in a script to set a command line argument for Mozilla Firefox:

```
baseState.setConnectionString("moz:firefoxOptions={\"args\":
[\"--devtools\"]};");
```

You can add the following code to the base state in a script to automatically download executables from Google Chrome to a specific folder:

```
baseState.setConnectionString(  
  "chromeOptions="  
  + "{ "  
  + "  \"prefs\": { "  
  + "  
  + "\"profile.default_content_setting_values.automatic_downloads\": "  
  + "1, "  
  + "  \"download.default_directory\": \"c:\\\\ "  
  + "\"Download\", "  
  + "  \"download.prompt_for_download\": false "  
  + "  } "  
  + "};");
```

You can add the following code to the base state in a script to disable the password manager from showing messages in Google Chrome:

```
baseState.setConnectionString(  
  "chromeOptions="  
  + "{ "  
  + "  \"args\": [\"--disable-save-password-bubble\"], "  
  + "  \"prefs\": { "  
  + "    \"profile.password_manager_enabled\": "  
  + "false, "  
  + "    \"credentials_enable_service\": false "  
  + "  } "  
  + "};");
```

Testing with Apple Safari on a Mac

This section describes how you can enhance your cross-browser test set by testing Apple Safari on Mac machines that are connected to a Windows machine on which Silk4J is installed.

Prerequisites for Testing with Apple Safari on a Mac

Before you can test with Apple Safari on a Mac, ensure that the following prerequisites are met:

- The Mac is connected as a remote location to a Windows machine, on which Silk4J is installed. For additional information, see *Editing Remote Locations*.
- If you are testing with Apple Safari 9, the SafariDriver, which is the WebDriver extension for Apple Safari that inverts the traditional client/server relationship and communicates with the WebDriver client using WebSockets, needs to be installed on the Mac. With Apple Safari 10.1, Safari features a built-in driver implementation.
- Java JDK is installed on the Mac.
- The information service is installed on the Mac. To get the files that are required for the information service, use the Silk Test installer. For additional information, see [Installing the Silk Test Information Service on a Mac](#).
- To run tests on Apple Safari, the user that has installed the information service needs to be logged in on the Mac.



Tip: Micro Focus recommends to set the Mac to automatically log in the correct user during startup. For additional information, see [Set your Mac to automatically log in during startup](#).

- To run unattended tests against Apple Safari on a Mac, adjust the following energy-related settings in the **Energy Saver** pane of the **System Preferences**:
 - Set **Turn display off after** to **Never**.
 - Check the **Prevent computer from sleeping automatically when the display is off** check box.



Note: You can use the **Silk Test Configuration Assistant** to easily configure such settings. To open the **Configuration Assistant** on a Mac, click on the Silk Test icon in the status menus and select **Configuration Assistant**.

- To run unattended tests against Apple Safari on a Mac, disable the screen saver.
 1. Navigate to **System Preferences > Desktop & Screen Saver**.
 2. Click the **Screen Saver** tab.
 3. Set **Start screen saver** to **Never**.



Note: You can use the **Silk Test Configuration Assistant** to easily configure such settings. To open the **Configuration Assistant** on a Mac, click on the Silk Test icon in the status menus and select **Configuration Assistant**.

- If you are testing with Apple Safari 10.1, enable the Safari developer menu. Choose **Safari > Preferences**, click **Advanced**, and check **Show develop menu in menu bar**.
- If you are testing with Apple Safari 10.1, enable remote automation. In the Safari developer menu, check **Allow Remote Automation**.
- When executing a test for the first time against Apple Safari 10.1, you need to provide a password.

Preparing Apple Safari for Testing

To test web applications on Apple Safari 10.1 or later, you can use the **Silk Test Configuration Assistant** to easily configure Apple Safari. To open the **Configuration Assistant** on a Mac, click on the Silk Test icon in the status menus and select **Configuration Assistant**. As an alternative, you can also perform the following preparation steps in addition to fulfilling the requirements listed in [Prerequisites for Testing with Apple Safari on a Mac](#):

1. Enable remote automation in Apple Safari, by opening the **Develop** menu and checking **Allow Remote Automation**.

The **Develop** menu is hidden by default. To open the menu:

 - a) In the **Safari** menu, choose **Preferences**.
 - b) In the **Preferences** window, select the **Advanced** tab.
 - c) Check the **Show Develop menu in menu bar** check box.
 - d) Close the **Preferences** window.
2. When running a test for the first time on Apple Safari, a dialog box appears, stating that the browser window is remotely controlled by an automated test. Click **Continue Session**.

For additional information on Apple Safari and Selenium WebDriver, see <https://webkit.org/blog/6900/webdriver-support-in-safari-10/>.

Installing the Silk Test Information Service on a Mac



Note: To install the information service on a Mac, you require administrative privileges on the Mac.

To create and execute tests on a Mac using Apple Safari or using iOS or Android devices, install the Silk Test information service (information service) on the Mac. Once the information service is installed and active, you can record and replay tests from a Silk4J client that is installed on a Windows machine.



Note: You cannot record on a Mac. To add a Mac as a test location to Silk4J, see *Editing Remote Locations* in the Silk4J documentation.

To install the information service on a Mac:

1. Read the information in the topic *Prerequisites for Testing with Apple Safari on a Mac* in the Silk4J documentation.
2. Ensure that a Java JDK is installed on the Mac.
3. If you want to test mobile applications on an iOS device, ensure that Xcode is installed on the Mac.

4. Access the information service setup file, `SilkTestInformationService<Version>-<Build Number>.pkg`.

- If you have downloaded the information service setup file while installing Silk Test, open the folder macOS in the Silk Test installation directory, for example `C:\Program Files (x86)\SilkTest\SilkTest`.
- If you have not downloaded the information service setup file while installing Silk Test, you can download the setup file from [Micro Focus SupportLine](#).

5. Copy the file `SilkTestInformationService<Version>-<Build Number>.pkg` to the Mac.

6. Execute `SilkTestInformationService<Version>-<Build Number>.pkg` to install the information service.

7. Follow the instructions in the installation wizard.

8. When asked for the password, provide the password of the currently signed in Mac user.

9. When Apple Safari opens and a message box asks whether to trust the SafariDriver, click **Trust**.



Note: If you want to test against Apple Safari 10 or prior on macOS 10.12 (Sierra) or prior, SafariDriver needs to be installed on the Mac. You can only install the SafariDriver if you are directly logged in to the Mac, and not connected through a remote connection.

10. To complete the installation, the installer logs the current Mac user out. To verify that the information service was installed correctly, log in to the Mac.

11. If you are installing the information service on a Mac with macOS Mojave (10.14) or later, you might have to enable additional automation permissions for Silk Test after logging in to the Mac.

If permissions need to be granted, Silk Test will automatically show request permission dialogs.

- a) Click **OK** to acknowledge the information dialog.
- b) Click **OK** in all sub-sequent request permission dialogs.



Important: If you do not enable these permissions for Silk Test, you will not be able to test web applications against Google Chrome or mobile applications on an iOS device or on a Simulator on this Mac. If by mistake you have clicked **Don't Allow** in one of the permission dialogs, open a terminal on the Mac and type the following command:

```
sudo tccutil reset AppleEvents
```

Then restart the Mac and accept the permission dialogs by clicking **OK**.

12. Click on the Silk Test icon in the top-right corner of the screen to see the available devices and browsers.



Tip: If the Silk Test icon does not appear, restart the Mac.

Limitations for Testing with Apple Safari

The following are the known limitations for testing with Apple Safari on a Mac:

- The following classes, interfaces, methods, and properties are currently not supported when testing web applications with Apple Safari on a Mac:
 - `BrowserApplication` class.
 - `clearCache` method
 - `closeOtherTabs` method
 - `closeTab` method
 - `existsTab` method
 - `getHorizontalScrollbar` method
 - `getNextCloseWindow` method
 - `getSelectedTab` method

- `getSelectedTabIndex` method
- `getSelectedTabName` method
- `getTabCount` method
- `getVerticalScrollbar` method
- `isActive` method
- `minimize` method
- `openContextMenu` method
- `openTab` method
- `restore` method
- `selectTab` method
- `setActive` method
- `windowState` property
- `BrowserWindow` class.
 - `acceptAlert` method
 - `dismissAlert` method
 - `getAlertText` method
 - `isAlertPresent` method
 - `mouseMove` method
 - `pressKeys` method
 - `pressMouse` method
 - `releaseKeys` method
 - `releaseMouse` method
- `IMoveable` class.
 - `getFocus` method.
- Silk4J does not support the **CMD** key for the `typeKeys` method.
- Silk4J does not support testing Apache Flex.
- Silk4J does not support testing iframes with a JavaScript source on Apple Safari.
- To test secure web applications over HTTPS on Apple Safari, ensure that any required server certificates are trusted.
- Silk4J does not provide native support for Apple Safari. You cannot test internal Apple Safari functionality. For example, in a test, you cannot change the currently displayed web page by adding text to the navigation bar. As a workaround, you can use API calls to navigate between web pages.
- Silk4J does not support JavaScript dialog API functions for Apple Safari. As a workaround, you could patch such functions so that they are ignored. For additional information, see <https://groups.google.com/forum/#!topic/selenium-developer-activity/qsovJw93g9c>.
- Silk4J does not support tabbing on Apple Safari.
- To test a multi window application, disable the Apple Safari pup-up blocker. To do so, start Apple Safari and navigate to **Safari Preferences > Security > Block pop-up window**.
- Silk4J does not support testing the dialog box for saving a password. To avoid this dialog box, start Apple Safari, navigate to **Safari Preferences > AutoFill**, and check the **User names and passwords** check box.
- Silk4J does not support properties in XPath expressions for Apple Safari. Only attributes are supported in XPath expressions.
- Silk4J does not support testing web applications which include a Content-Security-Policy HTTP header.
- With Apple Safari 10.1, Silk4J does not support navigating back in the browser.
- With Apple Safari 10.1, Silk4J does not support using control keys in the `typeKeys` method.
- With Apple Safari 10.1, Silk4J only supports dom actions in Frames and IFrames.
- With Apple Safari 10.1, Silk4J does not support navigating with Frames and IFrames.
- With Apple Safari 10.1, Silk4J does not support direct scrolling during recording. As a workaround, you could use the `executeJavaScript` method.

Running Multiple Apple Safari Tests at the Same Time

To execute a test on Apple Safari, you require a Mac that is connected to the Windows machine on which Silk Test is installed. If multiple Apple Safari want to execute tests on Apple Safari, these tests can be executed simultaneously on the same Mac.

 **Note:** Each test that is executed against Apple Safari on the Mac opens an individual instance of Apple Safari. Having too many instances of Apple Safari running simultaneously might reduce the performance of the Mac.

Uninstalling the Silk Test Information Service from a Mac

To uninstall the Silk Test information service (information service) from a Mac, for example if you no longer want to execute tests against Apple Safari on the Mac:

1. On the Mac, create a new shell file, for example `uninstallInfoService.sh`.
2. Type the following code into the new file:

```
#!/bin/sh

if launchctl list | grep com.borland.infoservice ; then
    launchctl unload /Library/LaunchAgents/com.borland.infoservice.plist
    echo "unloading Launch Daemon"
fi

if [ -d "/Applications/Silk" ]
then
    sudo rm -rf /Applications/Silk
fi

if [ -f "/Library/LaunchAgents/com.borland.infoservice.plist" ]
then
    sudo rm /Library/LaunchAgents/com.borland.infoservice.plist
fi

if [ -f "/usr/local/bin/ideviceinstaller" ]
then
    sudo rm /usr/local/bin/ideviceinstaller
fi

exit 0
```

3. In the command line, type `chmod +x uninstallInfoService.sh` to make the shell file executable.
4. Execute the shell file from the command line.

Testing with Google Chrome

This section describes how you can enhance your cross-browser test set by testing with Google Chrome.

Silk4J supports recording actions and replaying tests on Google Chrome.

- When starting to test on Google Chrome, with no running instance of Google Chrome open, Silk4J starts a new instance of Google Chrome. This new browser uses a temporary profile without add-ons and with an empty cache.
- When starting to test on an instance of Google Chrome, which is already running, Silk4J restarts Google Chrome with the same command line arguments that were used when the instance was initially started. This restart is required to enable the Silk4J automation support.
- When testing with Google Chrome, the Google Chrome instance is closed when shutting down the Open Agent or when starting to test another application outside Google Chrome.



Tip: If you want to execute an existing test script with Google Chrome, Micro Focus recommends that you use a base state or that you add a command to the test script to navigate to the URL.

Example 1

If the running instance of Google Chrome was initially started with the command `C:\Program Files (x86)\Google\Chrome\Application\chrome.exe www.borland.com`, Google Chrome opens to *www.borland.com* after the restart.

Example 2

If the running instance of Google Chrome was initially started with the command `C:\Program Files (x86)\Google\Chrome\Application\chrome.exe`, Google Chrome opens to *about:blank* after the restart.

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.



Note: To test a web application that is stored in the local file system, navigate to the `chrome://extensions` in Google Chrome and check the **Allow access to file URLs** check box for the **Silk Test Chrome Extension**.

Testing Google Chrome Extensions

You can use one of the following two approaches to test a Google Chrome extension (add-on) with Silk4J:

Install the extension as a .crx file when starting Google Chrome

To test a Google Chrome extension that is installed as a .crx file, add the following command line to the base state:

```
chrome.exe --load-extension=C:/myExtension/myExtension.crx
```



Note: You can only install a single extension in Google Chrome as a .crx file. To install multiple extensions in Google Chrome, use a comma separated list of .crx files. For example:

```
chrome.exe --load-extension=C:/myExtension/myExtension.crx,C:/myExtension2/myExtension2.crx
```

For information on adding command line arguments to a browser, see *Modifying the Base State*.

Add the extension to a profile

Add the extension to a Google Chrome user data directory and use that profile for testing. For additional information, see [Testing Google Chrome with User Data Directories](#).

Testing Google Chrome with User Data Directories

All changes that you make in Google Chrome, for example your home page, what toolbars you use, any saved passwords, and your bookmarks, are all stored in a special folder, which is called a user data directory.

With Silk4J, you can test Google Chrome user data directories by specifying the path to the user data directory in the base state of the application under test. The following command line includes the path to the profile:

```
chrome.exe "--user-data-dir=C:/Users/MyUser/AppData/Local/Google/Chrome/User Data"
```

To set the profile directory for our sample web application, you can use the following code:

```
BrowserBaseState baseState = new BrowserBaseState(BrowserType.GoogleChrome, "demo.borland.com\InsuranceWebExtJS");  
String myProfileDir = "--user-data-dir=C:\\temp\\SilkTest \"--profile-directory=my user\"";  
baseState.setCommandLineArguments(myProfileDir);  
desktop.executeBaseState(baseState);
```



Note: When Google Chrome is started by Silk4J, an empty user data directory is used. This ensures that the test starts at a clean state.

Limitations for Testing with Google Chrome

The following list lists the known limitations for playing back tests and recording locators with Google Chrome on a local Windows machine:

- Silk Test does not support testing child technology domains of the xBrowser domain with Google Chrome. For example Apache Flex or Microsoft Silverlight are not supported with Google Chrome.
- Silk4J does not support recording a test in an HTTP Basic Authentication dialog.
- 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 through the Alerts API.
- Silk4J does not support the `getFocus` method of the `IMoveable` class.
- Silk Test does not recognize opening the **Print** dialog box in Google Chrome by using the Google Chrome menu. To add opening this dialog box in Google Chrome to a test, you have to send **Ctrl+Shift**

+P using the `TypeKeys` method. Internet Explorer does not recognize this shortcut, so you have to first record your test in Internet Explorer, and then manually add pressing **Ctrl+Shift+P** to your test.

- Testing on multiple Google Chrome windows at the same time is only supported if the additional windows are opened from the initial Google Chrome window by the AUT itself. If the additional Google Chrome windows are opened manually, Silk4J does not recognize the elements on these Google Chrome windows. For example, Silk4J recognizes the elements in a Google Chrome window that is opened by clicking on a link or a button in the AUT during recording, but Silk4J does not recognize the elements in a Google Chrome window that was opened by pressing **CTRL+N** during recording.
- When using Internet Explorer to replay a test, you can use the following code to test `executeJavaScript`:

```
// Java code
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("function foo() { alert('Silk Test'); }");
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("foo();");
```

When replaying tests on Google Chrome, the scripts are not executed in the global context (window), but in a closure. Everything is executed within a function. The first `ExecuteJavaScript` call in the previous code sample will not work with Google Chrome, because the function `foo` is only available as long as the `ExecuteJavaScript` call lasts.

To replay the same test on Google Chrome, you can use the following function expression:

```
// Java code
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("window.foo = function() { alert('Silk Test'); }");
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("window.foo();");
```

The previous code samples will work in Silk4J. The code for the other Silk Test clients is similar. For additional information, refer to the documentation of the `ExecuteJavaScript` method in the Help of your Silk Test client.

- Parallel testing on Google Chrome does not work if the user data directory for Google Chrome is set through a group policy. As a workaround, ask your administrator to remove the registry key `HKEY_LOCAL_MACHINE\Software\Policies\Google\Chrome\UserDataDir` or `HKEY_CURRENT_USER\Software\Policies\Google\Chrome\UserDataDir`.

Limitations for Testing with Google Chrome on macOS

The following list lists the known limitations for playing back tests and recording locators with Google Chrome on macOS:

- Silk4J does not support the **CMD** key for the `typeKeys` method.
- Silk Test does not support testing child technology domains of the `xBrowser` domain with Google Chrome. For example Apache Flex or Microsoft Silverlight are not supported with Google Chrome.
- Silk4J does not support recording a test in an HTTP Basic Authentication dialog.
- 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 through the Alerts API.
- Silk4J does not support the `getFocus` method of the `IMoveable` class.
- Testing on multiple Google Chrome windows at the same time is not supported on macOS.
- Attaching to an already opened Google Chrome window on macOS is not supported.
- When using Internet Explorer to replay a test, you can use the following code to test `executeJavaScript`:

```
// Java code
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("function foo() { alert('Silk Test'); }");
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("foo();");
```

When replaying tests on Google Chrome, the scripts are not executed in the global context (window), but in a closure. Everything is executed within a function. The first `ExecuteJavaScript` call in the previous code sample will not work with Google Chrome, because the function `foo` is only available as long as the `ExecuteJavaScript` call lasts.

To replay the same test on Google Chrome, you can use the following function expression:

```
// Java code
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("window.foo = function() { alert('Silk Test'); }");
desktop.<BrowserWindow> find("//BrowserWindow")
    .executeJavaScript("window.foo();");
```

The previous code samples will work in Silk4J. The code for the other Silk Test clients is similar. For additional information, refer to the documentation of the `ExecuteJavaScript` method in the Help of your Silk Test client.

Testing with Mozilla Firefox

This section describes how you can enhance your cross-browser test set by testing with Mozilla Firefox.

Silk4J supports recording actions and replaying tests on Mozilla Firefox.

- When starting to test on Mozilla Firefox, with no running instance of Mozilla Firefox open, Silk4J starts a new instance of Mozilla Firefox. This new browser uses a temporary profile without add-ons and with an empty cache.
- When starting to test on an instance of Mozilla Firefox which is already running, Silk4J restarts Mozilla Firefox with the same command line arguments that were used when the instance was initially started. This restart is required to enable the Silk4J automation support.
- The Mozilla Firefox instance is closed when shutting down the Open Agent or when starting to test another application outside Mozilla Firefox.



Tip: If you want to execute an existing test script with Mozilla Firefox, Micro Focus recommends that you use a base state or that you add a command to the test script to navigate to the URL.

While recording against Mozilla Firefox, Mozilla Firefox opens external links in a new tab, instead of a new window. Disable the **OPT_FIREFOX_SINGLE_WINDOW_MODE** option in the **Browser** options to open external links in a new window.

Example 1

If the running instance of Mozilla Firefox was initially started with the command `C:\program files\Mozilla/firefox.exe www.borland.com`, Mozilla Firefox opens to *www.borland.com* after the restart.

Example 2

If the running instance of Mozilla Firefox was initially started with the command `C:\program files\Mozilla/firefox.exe`, Mozilla Firefox opens to *about:blank* after the restart.

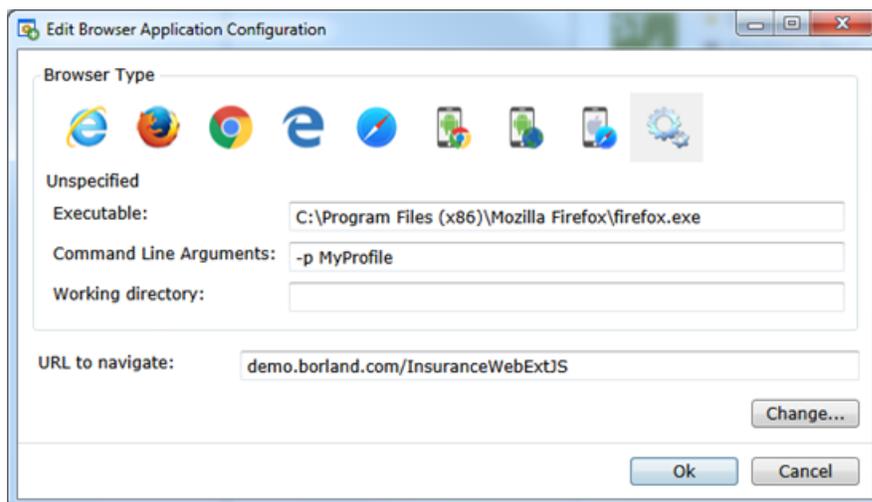
Testing Mozilla Firefox with Profiles

All changes that you make in Mozilla Firefox, for example your home page, what toolbars you use, any saved passwords, and your bookmarks, are all stored in a special folder, which is called a profile.

To test Mozilla Firefox profiles, you can specify either the name of the profile or the path to the profile as custom browser command line arguments in the **Edit Application Configurations** dialog box:

1. Select the project from which you want to test Mozilla Firefox profiles.

2. Click **Edit Application Configuration** in the menu. The **Edit Application Configuration** dialog box appears.
3. Uncheck the **Show 'Select Browser' dialog before record and playback** check box.
4. Click **Edit** to edit the existing **Browser: Firefox** application configuration.
5. Select **Custom** as the browser type.
6. Type the path to the Mozilla Firefox executable into the **Executable** field. For example `C:\Program Files (x86)\Mozilla Firefox\firefox.exe`.
7. Type the Firefox profile name or the path to the Firefox profile into the **Command Line Arguments** field. For example `-p myProfile` or `-profile C:/Temp`



Using the name of the profile

The following command line argument specifies the name of the profile:

```
-p myProfile
```

You can configure the named profile in the **Profile Manager**. To start the **Profile Manager**, type `firefox.exe -P` into a command window.

Using the path to the profile

The following command line argument specifies the path to the profile:

```
-profile C:/<path to profile folder>
```

 **Note:** If Mozilla Firefox is started by Silk4J, an empty profile is used. This ensures that the test starts at a clean state. If the user manually starts Mozilla Firefox, the default profile is used.

 **Note:** As profiles need to be deployed to the test machine are and have high memory consumption, you might face some issues when testing profiles. If you only want to change a few browser settings, you can use capabilities instead of profiles. For additional information, see [Setting Capabilities for Web-Driver Based Browsers](#).

Testing Mozilla Firefox Extensions

To test a Mozilla Firefox extension (add-on) with Silk4J, add the extension to a Mozilla Firefox profile and use that profile for testing. For additional information, see [Testing Mozilla Firefox with Profiles](#).

Limitations for Testing with Mozilla Firefox

The following limitations are known when testing web applications with Silk4J on Mozilla Firefox:

- Testing on multiple Mozilla Firefox windows at the same time is only supported if the additional windows are opened from the initial Mozilla Firefox window by the AUT itself. If the additional Mozilla Firefox windows are opened manually, Silk4J does not recognize the elements on these Mozilla Firefox windows. For example, Silk4J recognizes the elements in a Mozilla Firefox window that is opened by clicking on a link or a button in the AUT during recording, but Silk4J does not recognize the elements in a Mozilla Firefox window that was opened by pressing **CTRL+N** during recording.

- Silk4J does not support testing modal browser windows, which are windows that can be displayed with the `window.showModalDialog` command. These modal browser windows have been officially deprecated, and are disabled with Google Chrome 37 or later, while they are planned to no longer be supported in future versions of Mozilla Firefox. You can work around this issue by using low-level actions, for example native clicks with coordinates to click on an object or typekeys to fill out text fields.
- Silk4J does not support testing Silverlight with Mozilla Firefox.
- Silk4J does not support testing some browser dialogs, for example the **About** dialog, with Mozilla Firefox.
- Silk4J does not support testing `about : *` pages with Mozilla Firefox.
- Silk4J does not support recording a click on the **Print** button in Mozilla Firefox. To click on this button during replay, you can manually add a desktop click with coordinates to your test script. For example:


```
desktop.click(MouseButton.LEFT, printButton.getRect(true).getCenter());
```
- Silk Test does not provide native support for Mozilla Firefox. You cannot test internal Mozilla Firefox 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 through the Alerts API.
- Silk4J does not support recording locators on JavaScript alert boxes with Mozilla Firefox. Additionally, you cannot use the following methods to handle Javascript alert boxes with Mozilla Firefox:
 - `acceptAlert`
 - `dismissAlert`
 - `getAlertText`
 - `isAlertPresent`
- Silk4J does not support Java applets for Mozilla Firefox.
- Silk4J does not support properties in XPath expressions for Mozilla Firefox. Only attributes are supported in XPath expressions.
- Silk4J does not support the `getFocus` method of the `IMoveable` class.
- Silk Test does not support testing child technology domains of the `xBrowser` domain with Mozilla Firefox. For example Apache Flex or Microsoft Silverlight are not supported with Mozilla Firefox.
- With Mozilla Firefox, the following methods are not supported:
 - `pressKeys`
 - `releaseKeys`
- With Mozilla Firefox, native playback for the following is not supported:
 - Double-click.
 - Right and middle mouse button click.
- With Mozilla Firefox, the `domClick` method is not supported on controls that open an alert.

Limitations for Testing with Mozilla Firefox on macOS

The following limitations are known when testing web applications with Mozilla Firefox on macOS:

- Silk4J has been tested on macOS with Mozilla Firefox 54 or later.
- Testing on multiple Mozilla Firefox windows at the same time is only supported if the additional windows are opened from the initial Mozilla Firefox window by the AUT itself. If the additional Mozilla Firefox windows are opened manually, Silk4J does not recognize the elements on these Mozilla Firefox windows. For example, Silk4J recognizes the elements in a Mozilla Firefox window that is opened by clicking on a link or a button in the AUT during recording, but Silk4J does not recognize the elements in a Mozilla Firefox window that was opened by pressing **CTRL+N** during recording.
- Silk4J does not support testing modal browser windows, which are windows that can be displayed with the `window.showModalDialog` command. These modal browser windows have been officially deprecated, and are disabled with Google Chrome 37 or later, while they are planned to no longer be supported in future versions of Mozilla Firefox. You can work around this issue by using low-level actions, for example native clicks with coordinates to click on an object or typekeys to fill out text fields.

- Silk4J does not support testing Silverlight with Mozilla Firefox.
- Silk4J does not support testing some browser dialogs, for example the **About** dialog, with Mozilla Firefox.
- Silk4J does not support testing `about:*` pages with Mozilla Firefox.
- Silk4J does not support recording a click on the **Print** button in Mozilla Firefox. To click on this button during replay, you can manually add a desktop click with coordinates to your test script. For example:


```
desktop.click(MouseButton.LEFT, printButton.getRect(true).getCenter());
```
- Silk Test does not provide native support for Mozilla Firefox. You cannot test internal Mozilla Firefox 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 through the Alerts API.
- Silk4J does not support recording locators on JavaScript alert boxes with Mozilla Firefox. Additionally, you cannot use the following methods to handle Javascript alert boxes with Mozilla Firefox:
 - `acceptAlert`
 - `dismissAlert`
 - `getAlertText`
 - `isAlertPresent`
- Silk4J does not support Java applets for Mozilla Firefox on macOS.
- Silk4J does not support properties in XPath expressions for Mozilla Firefox. Only attributes are supported in XPath expressions.
- Silk4J does not support the `getFocus` method of the `IMoveable` class.
- Silk Test does not support testing child technology domains of the `xBrowser` domain with Mozilla Firefox. For example Apache Flex or Microsoft Silverlight are not supported with Mozilla Firefox.
- The following methods are not supported:
 - `pressKeys`
 - `releaseKeys`
- Native playback for the following is not supported:
 - Double-click.
 - Right and middle mouse button click.
- The `domClick` method is not supported on controls that open an alert.
- With Mozilla Firefox 55, uploading a file does not work. For additional information, see [File upload no longer works with geckodriver 0.18.0 and Firefox 55](#).

Testing with Microsoft Edge

This section describes how you can enhance your cross-browser test set by testing with Microsoft Edge.

Limitations for Testing with Microsoft Edge

The following are the known limitations for testing with Microsoft Edge:

- The following classes, interfaces, methods, and properties are currently not supported when testing web applications on Microsoft Edge:
 - `BrowserApplication` class.
 - `clearCache` method
 - `closeOtherTabs` method
 - `closeTab` method
 - `existsTab` method
 - `getHorizontalScrollbar` method
 - `getNextCloseWindow` method

- `getSelectedTab` method
- `getSelectedTabIndex` method
- `getSelectedTabName` method
- `getTabCount` method
- `getVerticalScrollbar` method
- `isActive` method
- `minimize` method
- `openContextMenu` method
- `openTab` method
- `restore` method
- `selectTab` method
- `setActive` method
- `windowState` method
- The following methods of the `BrowserWindow` class are not supported for Microsoft Edge versions prior to build 38.14393, the Microsoft Edge version for Microsoft Windows 10 Anniversary Update.
 - `pressKeys` method
 - `releaseKeys` method
- Silk4J does not automatically bring the browser into the foreground when recording actions against Microsoft Edge.
- When testing with Microsoft Edge, the rectangle for the `BrowserApplication` is not absolute.
- Silk4J does not support testing Apache Flex.
- Silk4J does not provide native support for Microsoft Edge. You cannot test internal Microsoft Edge functionality. For example, in a test, you cannot change the currently displayed web page by adding text to the navigation bar. As a workaround, you can use API calls to navigate between web pages.
- Silk4J does not support handling alerts and similar dialog boxes for Microsoft Edge.
- Image clicks are only supported for Microsoft Edge Threshold 2 (build 25.10586) or later. If you are testing a web application on a prior version of Microsoft Edge, you can only use image verifications.
- Silk4J does not support tabbing on Microsoft Edge. Tabs are recognized as windows.
- When testing web applications on Microsoft Edge, Silk4J cannot locate meta-tags which include the `http-equiv` attribute. For example, Silk4J cannot locate the following meta-tag:


```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```
- With Microsoft Edge, Silk4J does not support directly reading the `currentStyle` attribute of a DOM element. You can use the `getCssStyle` method of the `DomElement` class to retrieve the computed CSS style with the specified style name.
- When starting the interaction with a web application on Microsoft Edge, Silk4J closes any open instance of Microsoft Edge and starts a new browser. This new browser uses a temporary profile without add-ons and with an empty cache. This instance of Microsoft Edge is closed when shutting down the Open Agent or when starting to test another application outside Microsoft Edge.
- With Microsoft Edge, Silk4J does not recognize the `textContent` attribute while recording actions or locators. However, you can use the `textContent` attribute in object maps and when replaying a test on Microsoft Edge.
- Silk4J does not support properties in XPath expressions for Microsoft Edge. Only attributes are supported in XPath expressions.
- Silk4J does not support the `getFocus` method of the `IMoveable` class.
- Silk4J does not support testing web applications which include a Content-Security-Policy HTTP header.

Responsive Web Design Testing

Desktop web applications which are built based upon responsive web design might change their appearance in response to the size of the screen or web browser in which these applications are displayed.

Choosing the appropriate size of the replay window might have significant impact on the stability of such tests.

Silk4J enables you to specify the exact size of the browser window in the following situations:

- When you create a new project for a web application.
- When you record a new test.
- When you record actions into an existing test.
- When you replay a test against a web application.

You can use the following settings to specify the size of the browser window:

- The **Browser size** list contains a mix of predefined and custom browser window sizes, enabling you to select the size on which you want to test.
- The **Orientation** list enables you to select whether you want the browser window to use landscape or portrait orientation.
- Click **Edit Browser Sizes** to add custom browser sizes to the **Browser size** list or to specify which browser sizes are displayed in the list.
 - To add a new custom browser size to the list, click **Add Browser Size**.
 - To exclude a size from the list, uncheck the corresponding check box.
 - To add the visual breakpoints of a specific web application to the **Browser size** list, click **Detect Visual Breakpoints**.

When replaying a test against a web application from the command line or from Silk Central, you can specify the size of the browser viewport by setting the `silktest.browserViewportSize` environment variable. You can either specify the name of a browser from the **Browser Size** list or a specific size.

You can change the size of the browser directly in a test script, if `import com.microfocus.silktest.jtf.mobile.common.types.DisplayOrientation;` is defined in the test script:

```
BrowserWindow window = desktop.<BrowserWindow>find("//BrowserApplication//  
BrowserWindow");  
window.setViewportName("Google Pixel 2", DisplayOrientation.LANDSCAPE);  
window.setViewportSize(800, 300);
```

Example: Setting the browser size for automated replay by using the name

The following code sample sets the browser size to SVGA (800, 600) by using the SVGA entry of the **Browser size** list:

```
SET silktest.browserViewportSize=name=SVGA;orientation=landscape
```

Example: Setting the browser size for automated replay by using the width and height

The following code sample sets the browser size to SVGA (800, 600) by using the `width` and `height` parameters:

```
SET  
silktest.browserViewportSize=width=800;height=600;orientation=la  
ndscape
```

Detecting Visual Breakpoints

Before detecting the visual breakpoints in a responsive web application, ensure that Mozilla Firefox 56 or later or Google Chrome 60 or later is installed on the machine on which Silk4J is running.

Many web applications that are implemented with responsive web design techniques change their layout in response to the size of the browser or device in which they are displayed. The specific resolutions on which the layout changes are called *visual breakpoints*.

Silk4J supports testing such applications by detecting the visual breakpoints, and by allowing you to resize the recording window to the specific size of such a visual breakpoint.

Silk4J enables you to specify the exact size of the browser window in the following situations:

- When you create a new project for a web application.
- When you record a new test.
- When you record actions into an existing test.
- When you replay a test against a web application.

To find the visual breakpoints in a web application, and to display the corresponding resolutions in the **Browser size** list, perform the following actions:

1. Click **Edit Browser Sizes**. The **Edit Browser Sizes** dialog appears.
2. Click **Detect Visual Breakpoints**. If no URL for the web application is specified in an application configuration or base state, the **Visual Breakpoint Detection URL** dialog appears.
3. If no URL for the web application has been specified in an application configuration or base state, the **Visual Breakpoint Detection URL** dialog appears. Type the URL into the text field and click **OK**. Silk4J detects all visual breakpoints for the web application and adds them to the **Browser sizes** list.
4. Click **OK** to close the **Edit Browser Sizes** dialog.

You can now select any of the visual breakpoints as the size of the browser window or mobile device for testing.

Improving iframe Performance

Replaying tests against web applications that contain many iframes on browsers other than Internet Explorer might sometimes be slow. This topic provides some performance optimization suggestions when using the following browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari
- A supported mobile browser on a mobile device or on an emulator.

If you are facing such performance issues, you could try to apply some performance optimizations to your test scripts.

For example, assume that you are testing against a web application with an iframe structure that looks similar to the following:

```
BrowserWindow
  iframe name=1
    iframe name=2
    iframe name=3
    iframe name=4
    iframe name=5
  iframe name=6
    iframe name=7
    iframe name=8
    iframe name=9
```

You could try to apply one or more of the following performance optimizations:

Using the Whitelist for iframe support to enable only the iframes that you are interested in

For example, in the previous iframe structure, assume you only want to interact with elements in the iframe with the name 8. A locator for such an element would look like the following:

```
//BrowserApplication//BrowserWindow//IFRAME[@name='6']//  
IFRAME[@name='8']//input[@name='username']
```



Note: All supported browsers except Internet Explorer include iframes in the locator.

To interact with the elements from the iframe with the name 8, you need to also include all parent iframes in the **Whitelist for iframe support**. For this example, type `name:6`, `name:8` into the **Whitelist for iframe support** field.

Using the Blacklist for iframe support to disable the iframes that you are not interested in

You can use the **Blacklist for iframe support** to disable iframes that you are not interested in, for example iframes for ads.

For example, in the previous iframe structure, if you know that you will never test elements in the iframe with the name 1 or in its child iframes, type `name:1` into the **Blacklist for iframe support** field.

Completely disabling iframe support

You can completely disable the iframe support by unchecking the option **OPT_XBROWSER_ENABLE_IFRAME_SUPPORT**.

For example, if all iframes in your application are only used for ads, and you do not want to test any of these iframes, you can use this option.

Adapt locators to reduce the number of iframe searches

To enhance the replay performance of find actions, you can adapt the locators to reduce the number of iframe searches. Applying this performance optimization might work very well in combination with using the **Blacklist for iframe support** or **Whitelist for iframe support**.

For example, in the previous iframe structure, assume you only want to interact with elements in the iframe with the name 8. You can adapt the Xpath locators or the object map entries to reduce the number of iframe searches.

The following sample script searches multiple times for the iframe with the name 8.

```
desktop.<DomElement>find("//BrowserApplication//BrowserWindow//  
IFRAME[@name='6']//IFRAME[@name='8']//  
input[@name='username']").typeKeys("my user name");  
desktop.<DomElement>find("//BrowserApplication//BrowserWindow//  
IFRAME[@name='6']//IFRAME[@name='8']//  
input[@name='password']").typeKeys("top secret");  
desktop.<DomElement>find("//BrowserApplication//BrowserWindow//  
IFRAME[@name='6']//IFRAME[@name='8']//  
input[@name='loginButton']").click();
```

You can adapt the locators in the previous sample script to only search once for the iframe with the name 8.

```
DomElement iframe = desktop.<DomElement>find("//  
BrowserApplication//BrowserWindow//IFRAME[@name='6']//  
IFRAME[@name='8']");  
iframe.<DomElement>find("//  
input[@name='username']").typeKeys("my user name");  
iframe.<DomElement>find("//  
input[@name='password']").typeKeys("top secret");  
iframe.<DomElement>find("//input[@name='loginButton']").click();
```

Testing Additional Browser Versions

This topic describes how you can test on additional versions of WebDriver-based browsers, which are not automatically included in your Silk4J version.



Note: Silk4J should be able to automatically support the newest version of a browser. However, if the browser vendor has introduced changes which require an update to Silk4J, the procedure described in this topic might not enable testing on the new browser version.

The functionality described in this topic is supported for the following browsers:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox

1. Download the appropriate driver for the browser version that you want to test.

- For Google Chrome, you can download additional ChromeDriver versions from [Downloads - ChromeDriver](#).
- For Mozilla Firefox, you can download additional geckodriver versions from [Releases - mozilla/geckodriver](#).
- For Microsoft Edge, you can download additional Microsoft WebDriver versions from [WebDriver - Microsoft Edge](#).

2. In the Silk4J installation folder, navigate to the folder `\ng\WebDrivers\`.

3. Open the folder that corresponds to the operating system on which you want to use the new browser version:

- For Microsoft Windows, open the folder `windows`.
- For macOS, open the folder `osx64`.

4. Open the appropriate folder for the browser.

- For Google Chrome, open `Chrome`.
- For Mozilla Firefox, open `Gecko`.
- For Microsoft Edge, open `Edge`.

5. Create a new folder for the new driver version.

For example, if the driver is ChromeDriver 2.26, create the new folder `2.26`.

6. Extract the downloaded driver into the new folder.

7. In the Silk4J installation folder, navigate to the folder `\ng\WebDrivers\Common\Config\`.

8. Open the appropriate folder for the browser.

- For Google Chrome, open `Chrome`.
- For Mozilla Firefox, open `Gecko`.
- For Microsoft Edge, open `Edge`.

9. Open the properties file in a text editor.

For example, for Google Chrome, open `Chrome.properties`.

10. Add the new browser version and the new driver version as follows:

```
<browser version>=<driver version>
```

For example, if you want to test on Google Chrome 53, you require ChromeDriver 2.26 and you have to add the following line to the `Chrome.properties` file:

```
53=2.26
```

11. Save the properties file.

Cross-Browser Testing: Frequently Asked Questions

This section includes questions that you might encounter when testing your Web application on various browsers.

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']
```

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.

How Can I Maximize the Browser Window when Starting to Test?

To maximize a browser inside a test script, for example when starting to test, you can use the `maximize` method of the `BrowserApplication` class.

To maximize your browser, add the following to your test script:

```
desktop.<BrowserApplication> find("//  
BrowserApplication").maximize();
```

How can I scroll in a browser?

Silk4J provides the following ways to scroll controls in a browser into view during replay:

executeJavaScript method (DomElement)

Use the `scrollIntoView` method to scroll a specific DOM element into the visible area of the browser window.

executeJavaScript method (BrowserWindow)

Use the `executeJavaScript` method to scroll the entire page up or down by a specified range.

Examples

The following command scrolls one page down:

```
browserWindow.executeJavaScript("window.scrollTo(0, window.innerHeight);");
```

The following command scrolls down 100 pixels:

```
browserWindow.executeJavaScript("window.scrollTo(0, 100);");
```

The following command scrolls up 100 pixels:

```
browserWindow.executeJavaScript("window.scrollTo(0, -100);");
```

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 tested versions, refer to the [Release Notes](#).

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.

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

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

Internet Explorer 8 or earlier

```
wDomElement.GetProperty("currentStyle:fontName")
```

**All other browsers, for example
Internet Explorer 9 or later and
Mozilla Firefox**

```
wDomElement.GetProperty("currentStyle:font-name")
```

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 `textContent` instead.

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.

`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.

Logging Output of My Application Contains Wrong Timestamps

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

My Test Script Hangs After Navigating to a New Page

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

Recorded an Incorrect Locator

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

Rectangles Around Elements in Internet Explorer are Misplaced

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

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

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

- Only records a `MoveMouse` action if the mouse stands still for a specific time.
- Only records `MoveMouse` actions if it observes activity going on after an element was hovered over. In some situations, you might need to add some manual actions to your script.
- Silk4J supports recording mouse moves only for Web applications, Win32 applications, and Windows Forms applications. Silk4J does not support recording mouse moves for child technology domains of the xBrowser technology domain, for example Apache Flex and Swing.

What is the Difference Between `textContent`, `innerText`, and `innerHTML`?

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

Code	Returned Value
<pre>browser.DomElement("//div[@id='mylinks']").GetProperty("innerHTML")</pre>	<pre>This is my link collection: Bye bye Borland Welcome to Micro Focus </pre>



Note: In Silk Test 13.5 or later, whitespace in texts, which are retrieved through the `textContent` property of an element, is trimmed consistently across all supported browsers. For some browser versions, this whitespace handling differs to Silk Test versions prior to Silk Test 13.5. You can re-enable the old behavior by setting the `OPT_COMPATIBILITY` option to a version lower than 13.5.0. For example, to set the option to Silk Test 13.0, type the following into your script:

```
desktop.setOption("OPT_COMPATIBILITY", "13.0.0");
```

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:

- When recording a script for cross-browser testing, Micro Focus recommends using Google Chrome, Mozilla Firefox, or Microsoft Edge, as a script recorded with Silk4J against Internet Explorer might slightly differ in comparison to a script recorded on one of the other browsers.
- Different attribute values. For example, colors in Internet Explorer are returned as "# FF0000" and in Mozilla Firefox as "rgb(255,0,0)".
- Different attribute names. For example, the font size attribute is called "fontSize" in Internet Explorer 8 or earlier and is called "font-size" in all other browsers, for example Internet Explorer 9 or later and Mozilla Firefox.
- Some frameworks may render different DOM trees.

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.

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.

Why Are Clicks Recorded Differently in Internet Explorer 10?

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

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

Why Do I Get an Invalidated-Handle Error?

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

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

When testing a web application, the reason for this problem might be 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.

As a best practice for all applications, Micro Focus recommends 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.

Starting a Browser from a Script

Instead of selecting a browser for replay at the start of a test, you might require to start a specific browser out of the test script during replay.

Using the `BrowserBaseState` class

Using the `BrowserBaseState` class to start a browser out of a test script ensures that the browser that is specified by the `Executable` property is running and ready for testing. The base state additionally navigates to the URL that is specified by the `Url` property and brings the browser to the front.

The following code sample uses the `BrowserBaseState` class to start Internet Explorer.

Using multiple instances of a browser

If you have more than one browser window or tabs open, Silk4J handles each browser window or tab as a distinct object with a unique locator. The locators are indexed, for example `WebBrowser`, `WebBrowser[1]`, `WebBrowser[2]`, and so on.

Finding Hidden Input Fields

Hidden input fields are HTML fields for which the tag includes `type="hidden"`. To enable a find to locate hidden input fields, you can use the `OPT_XBROWSER_FIND_HIDDEN_INPUT_FIELDS` option. The default value of the option is `TRUE`.

```
desktop.setOption(CommonOptions.OPT_XBROWSER_FIND_HIDDEN_INPUT_FIELDS, true);
```

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']
```

Custom Attributes for Web Applications

HTML defines a common attribute `ID` that can represent a stable identifier. By definition, the `ID` uniquely identifies an element within a document. Only one element with a specific `ID` can exist in a document.

However, in many cases, and especially with AJAX applications, the `ID` is used to dynamically identify the associated server handler for the HTML element, meaning that the `ID` changes with each creation of the Web document. In such a case the `ID` is not a stable identifier and is not suitable to identify UI controls in a Web application.

A better alternative for Web applications is to introduce a new custom HTML attribute that is exclusively used to expose UI control information to Silk4J.

Custom HTML attributes are ignored by browsers and by that do not change the behavior of the AUT. They are accessible through the DOM of the browser. Silk4J allows you to configure the attribute that you want to use as the default attribute for identification, even if the attribute is a custom attribute of the control class. To set the custom attribute as the default identification attribute for a specific technology domain, click **Silk4J > Edit Options > Custom Attributes** and select the technology domain.

The application developer just needs to add the additional HTML attribute to the Web element.

Original HTML code:

```
<A HREF="http://abc.com/control=4543772788784322..." <IMG  
src="http://abc.com/xxx.gif" width=16 height=16> </A>
```

HTML code with the new custom HTML attribute *AUTOMATION_ID*:

```
<A HREF="http://abc.com/control=4543772788784322..."  
AUTOMATION_ID = "AID_Login" <IMG src="http://abc.com/xxx.gif"  
width=16 height=16> </A>
```

When configuring the custom attributes, Silk4J uses the custom attribute to construct a unique locator whenever possible. Web locators look like the following:

```
...//DomLink[@AUTOMATION_ID='AID_Login']
```

Example: Changing ID

One example of a changing ID is the Google Widget Toolkit (GWT), where the ID often holds a dynamic value which changes with every creation of the Web document:

```
ID = 'gwt-uid-<nnn>'
```

In this case `<nnn>` changes frequently.

Limitations for Testing on Microsoft Windows 8 and Microsoft Windows 8.1

The following list lists the known limitations for testing on Microsoft Windows 8 and Microsoft Windows 8.1:

- Silk Test does not support testing Universal Windows Platform (UWP) apps (also known as Windows Store apps or Metro-style apps) on Microsoft Windows 8 and Microsoft Windows 8.1.

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 Apache 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, except for mobile applications, where the attribute names are case insensitive. Attribute values are by default case insensitive, but you can change the default setting like any other option. The locator attributes support the wildcards ? and *.

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

Attributes for Java AWT/Swing Applications

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

Supported attributes for Java AWT/Swing include:

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

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

Attributes for Java SWT Applications

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

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes

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

Attributes for SAP Applications

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

Supported attributes for SAP include:

- automationId
- caption

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

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

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

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

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

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

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

Attribute Type	Description	Example
automationId	An identifier that is provided by the developer of the application under test. The Visual Studio designer automatically assigns an <i>automationId</i> 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 <i>automationId</i> or <i>name</i> attribute instead of the <i>caption</i> .	//SLButton[@caption="Ok"]
className	The simple .NET class name (without namespace) of the Silverlight control. Using the <i>className</i> 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
<code><Button>Ok</Button></code>	<code>//SLButton[@caption="Ok"]</code>
<code><Button Name="okButton">Ok</Button></code>	<code>//SLButton[@automationId="okButton"]</code>
<code><Button AutomationProperties.AutomationId="okButton">Ok</Button></code>	<code>//SLButton[@automationId="okButton"]</code>

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

Locator Attributes for Identifying Controls with UI Automation

The supported locator attributes for controls in Windows-based applications that have implemented UI Automation provider interfaces include:

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



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

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

To identify controls in Windows-based applications that have implemented UI Automation provider interfaces within 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 `//UIAButton[@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 <i>automationId</i> to every control that is created with the designer. The application developer uses this ID to identify the control in the application code.	// UIAButton[@automationId="okButton"]
caption	The text that the control displays. When testing a localized application in multiple languages, use the <i>automationId</i> or <i>name</i> attribute instead of the <i>caption</i> .	//UIAButton[@caption="Ok"]
className	The class name (without namespace) of the UI Automation control. Using the <i>className</i> attribute can help to identify a custom control that is derived from a standard UI Automation control that Silk4J recognizes.	// UIAButton[@className='MyCustomButton']
name	The name of a control. Can be provided by the developer of the application under test.	// UIAButton[@name="okButton"]

During recording, Silk4J creates a locator for a UI Automation 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.

To find out which additional custom attributes you could use for the UI Automation controls in your AUT, you can use the **Verify Properties** dialog box. To do so, hover the mouse cursor over a UI Automation control during recording, and click **Ctrl+Alt**. You can then see which properties are available for the control. For example, for some applications, the attribute `value` is useful.

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 <i>Dynamic Locator Attributes</i> .

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

Attributes for Web Applications

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

Supported attributes for Web applications include:

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

 **Note:** Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

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

Uses the following locator:

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

Attributes for Windows Forms Applications

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

Supported attributes for Windows Forms applications include:

- automationid

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

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

Attributes for Windows Presentation Foundation (WPF) Applications

Supported attributes for WPF applications include:

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

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

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

Object Recognition

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

```
WPFButton[@automationId='okButton']"
```

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

Attribute Type	Description	Example
automationId	An ID that was provided by the developer of the test application.	//WPFButton[@automationId='okButton']"
name	The name of a control. The Visual Studio designer automatically assigns a name to every control that is created with the designer. The application developer uses this name to identify the control in the application code.	//WPFButton[@name='okButton']"
caption	The text that the control displays. When testing a	//WPFButton[@automationId='Ok']"

Attribute Type	Description	Example
	localized application in multiple languages, use the <i>automationId</i> or <i>name</i> attribute instead of the <i>caption</i> .	
<i>className</i>	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.	<code>//WPFButton[@className='MyCustomButton']"</code>

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

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

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

Attributes for Windows API-based Client/Server Applications

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

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

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

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

Dynamic Locator Attributes

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

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

Example

If you want to identify the button that has the user input focus in a Silverlight application, you can type:

```
browser.Find("//SLButton[@IsKeyboardFocused=true] ")
```

or alternatively

```
Dim button = dialog.SLButton("@IsKeyboardFocused=true")
```

This works because Silk4J exposes a property called `IsDefault` for the Silverlight button control.

Example

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

```
Dim button = browser.Find("//SLButton[@FontSize=12] ")
```

or alternatively

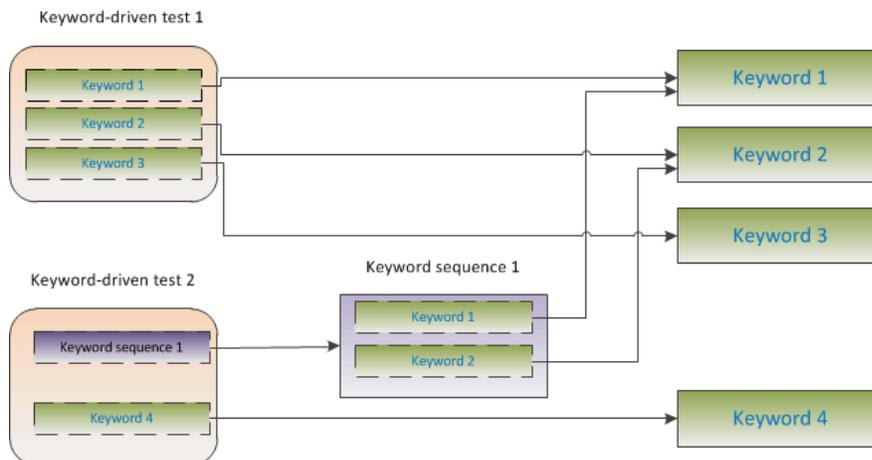
```
Dim button = browser.SLButton("@FontSize=12")
```

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

Keyword-Driven Tests

Keyword-driven testing is a software testing methodology that separates test design from test development and therefore allows the involvement of additional professional groups, for example business analysts, in the test automation process. Silk Central and Silk Test support the keyword-driven testing methodology and allow a very close collaboration between automation engineers and business analysts by having automation engineers develop a maintainable automation framework consisting of shared assets in the form of keywords in Silk Test. These keywords can then be used by business analysts either in Silk Test to create new keyword-driven tests or in Silk Central to convert their existing manual test assets to automated tests or to create new keyword-driven tests.

- A *keyword-driven test* is an executable collection of keywords. A keyword-driven test can be played back just like any other test.
- A *keyword sequence* is a keyword that is a combination of other keywords. Keyword sequences bundle often encountered combinations of keywords into a single keyword, enabling you to reduce maintenance effort and to keep your tests well-arranged.
- A *keyword* is a defined combination of one or more actions on a test object. The implementation of a keyword can be done with various tools and programming languages, for example Java or .NET.



There are two phases required to create keyword-driven tests:

1. Designing the test.
2. Implementing the keywords.

For a complete list of the record and replay controls available for keyword-driven testing, see the `com.borland.silk.keyworddriven.annotations` package in the *API Reference*.

Advantages of Keyword-Driven Testing

The advantages of using the keyword-driven testing methodology are the following:

- Keyword-driven testing separates test automation from test case design, which allows for better division of labor and collaboration between test engineers implementing keywords and subject matter experts designing test cases.
- Tests can be developed early, without requiring access to the application under test, and the keywords can be implemented later.
- Tests can be developed without programming knowledge.

- Keyword-driven tests require less maintenance in the long run. You need to maintain the keywords, and all keyword-driven tests using these keywords are automatically updated.
- Test cases are concise.
- Test cases are easier to read and to understand for a non-technical audience.
- Test cases are easy to modify.
- New test cases can reuse existing keywords, which amongst else makes it easier to achieve a greater test coverage.
- The internal complexity of the keyword implementation is not visible to a user that needs to create or execute a keyword-driven test.

Keywords

A *keyword* is a defined combination of one or more actions on a test object. The implementation of a keyword can be done with various tools and programming languages, for example Java or .NET. In Silk4J, a keyword is an annotated test method (`@Keyword`). Keywords are saved as keyword assets.

You can define keywords and keyword sequences during the creation of a keyword-driven test and you can then implement them as test methods. You can also mark existing test methods as keywords with the `@Keyword` annotation. In Java, keywords are defined with the following annotation:

```
@Keyword("keyword_name")
```

A *keyword sequence* is a keyword that is a combination of other keywords. Keyword sequences bundle often encountered combinations of keywords into a single keyword, enabling you to reduce maintenance effort and to keep your tests well-arranged.

A keyword or a keyword sequence can have a combined total of 20 input and output parameters. Any parameter of the test method that implements the keyword is a parameter of the keyword. To specify a different name for a parameter of a keyword, you can use the following:

```
// Java code
@Argument("parameter_name")
```

By default a parameter is an input parameter in Silk4J. To define an output parameter, use the class `OutParameter`.



Note: To specify an output parameter for a keyword in the **Keyword-Driven Test Editor**, use the following annotation:

```
${parameter_name}
```

In the **Keyword-Driven Test Editor**, you can use the same annotation to use an output parameter of a keyword as an input parameter for other keywords.

Example

A test method that is marked as a keyword can look like the following:

```
// Java code
@Keyword("Login")
public void login(){
    ... // method implementation
}
```

or

```
// Java code
@Keyword(value="Login", description="Logs in with the given
name and password.")
public void login(@Argument("UserName") String userName,
    @Argument("Password") String password,
    @Argument("Success") OutParameter success) {
```

```
... // method implementation  
}
```

where the keyword logs into the application under test with a given user name and password and returns whether the login was successful. To use the output parameter as an input parameter for other keywords, set the value for the output parameter inside the keyword.



Note: If you are viewing this help topic in PDF format, this code sample might include line-breaks which are not allowed in scripts. To use this code sample in a script, remove these line-breaks.

- The keyword name parameter of the `Keyword` annotation is optional. You can use the keyword name parameter to specify a different name than the method name. If the parameter is not specified, the name of the method is used as the keyword name.
- The `Argument` annotation is also optional. If a method is marked as a keyword, then all arguments are automatically used as keyword arguments. You can use the `Argument` annotation to specify a different name for the keyword argument, for example `UserName` instead of `userName`.

Creating a Keyword-Driven Test in Silk4J

Use the **Keyword-Driven Test Editor** to combine new keywords and existing keywords into new keyword-driven tests. New keywords need to be implemented as automated test methods in a later step.

1. Click **Silk4J > New Keyword-Driven Test**. The **New Keyword-Driven Test** dialog box opens.
2. Type a name for the new test into the **Name** field.
3. Select the project in which the new test should be included.

By default, if a project is active, the new test is created in the active project.



Note: To optimally use the functionality that Silk4J provides, create an individual project for each application that you want to test, except when testing multiple applications in the same test.

4. Click **Finish** to save the keyword-driven test.
5. Click **No** to create an empty keyword-driven test. The **Keyword-Driven Test Editor** opens.
6. Perform one of the following actions:
 - To add a new keyword, type a name for the keyword into the **New Keyword** field.
 - To add an existing keyword, expand the list and select the keyword that you want to add.
7. Press **Enter**.
8. Repeat the previous two steps until the test includes all the keywords that you want to execute.
9. Click **File > Save**.

Continue with implementing the keywords or with executing the test, if all keywords are already implemented.

Recording a Keyword-Driven Test in Silk4J

Before you can create a keyword-driven test in Silk4J, you have to select a project.

To record a single keyword, see [Recording a Keyword](#).

To record a new keyword-driven test:

1. Click **Silk4J > New Keyword-Driven Test**. The **New Keyword-Driven Test** dialog box opens.
 2. Type a name for the new test into the **Name** field.
 3. Select the project in which the new test should be included.
By default, if a project is active, the new test is created in the active project.
-  **Note:** To optimally use the functionality that Silk4J provides, create an individual project for each application that you want to test, except when testing multiple applications in the same test.
4. Click **Finish** to save the keyword-driven test.
 5. Click **Yes** to start recording the keyword-driven test. The **Record Keyword-Driven Test** dialog box opens.
 6. If you have not set an application configuration for the current project, select the tab that corresponds to the type of application that you are testing:
 - If you are testing a standard application that does not run in a browser, select the **Windows** tab.
 - If you are testing a web application or a mobile web application, select the **Web** tab.
 - If you are testing a native mobile application, select the **Mobile** tab.
 7. To test a standard application, select the application from the list.
 8. To test a web application or a mobile web application, if you have not set an application configuration for the current project, select one of the installed browsers or mobile browsers from the list.
 - a) Specify the web page to open in the **Enter URL to navigate** text box. If an instance of the selected browser is already running, you can click **Use URL from running browser** to record against the URL currently displayed in the running browser instance. For the tutorial, select **Internet Explorer** and specify <http://demo.borland.com/InsuranceWebExt.JS/> in the **Enter URL to navigate** text box.
 - b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
 - c) *Optional:* Select an **Orientation** for the browser window.
 - d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.
 9. To test a native mobile application (app) if you have not set an application configuration for the current project:
 - a) Select the mobile device, on which you want to test the app, from the list.
 - b) Select the native mobile application.
 - If you want to install the app on the mobile device or emulator, click **Browse** to select the app file or enter the full path to the app file into the **App file** text field. Silk4J supports HTTP and UNC formats for the path.
 - If you want to use an app that is already installed on an Android device, select the app from the **Package/Activity** list or specify the package and the activity in the **Package/Activity** field.
 - If you want to use an app that is already installed on an iOS device, specify the **Bundle ID**.
 - If you want to use an app that is available in Mobile Center, specify the **App identifier**.
 10. If you have set an application configuration for the current project and you are testing a web application, the **Select Browser** dialog box opens:
 - a) Select the browser.
 - b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
 - c) *Optional:* Select an **Orientation** for the browser window.
 - d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.

11. Depending on the dialog that is open, perform one of the following:

- In the **Select Application** dialog box, click **OK**.
- In the **Select Browser** dialog box, click **Record**.

12. In the application under test, perform the actions that you want to include in the first keyword.

For information about the actions available during recording, see *Actions Available During Recording*.

13. To specify a name for the keyword, hover the mouse cursor over the keyword name in the **Recording** window and click **Edit**.



Note: Silk4J automatically adds the keyword *Start application* to the start of the keyword-driven test. In this keyword, the applications base state is executed to enable the test to replay correctly. For additional information on the base state, see *Base State*.

14. Type a name for the keyword into the **Keyword name** field.

15. Click **OK**.

16. To record the actions for the next keyword, type a name for the new keyword into the **New keyword name** field and click **Add**. Silk4J records any new actions into the new keyword.

17. Create new keywords and record the actions for the keywords until you have recorded the entire keyword-driven test.

18. Click **Stop**. The **Record Complete** dialog box opens.

19. *Optional:* In the **Package** text box, specify the package name.

For example, type: `com.example`.

To use an existing package, click **Select** and select the package that you want to use.

20. In the **Test class** text box, specify the name for the test class.

For example, type: `AutoQuoteInput`.

To use an existing class, click **Select** and select the class that you want to use.

21. Click **OK**.

Silk4J creates the new keyword-driven test with all recorded keywords.

Setting the Base State for a Keyword-Driven Test in Silk4J

When you execute a keyword-driven test with Silk4J and the keyword-driven test calls a base state keyword, Silk4J starts your AUT from the base state.

During the recording of a keyword-driven test, Silk4J searches in the current project for a base state keyword, which is a keyword for which the `isBaseState` property is set to `true`.

- If a base state keyword exists in the current project, Silk4J inserts this keyword as the first keyword of the keyword-driven test.
- If there is no base state keyword in the project, Silk4J creates a new base state keyword with the name *Start application* and inserts it as the first keyword of the keyword-driven test.

To manually mark a keyword as a base state keyword, add the `isBaseState` property to the `Keyword` annotation, and set the value of the property to `true`:

```
@Keyword(value = "Start application", isBaseState = true)
public void start_application() {
    // Base state implementation
}
```

Implementing a Keyword in Silk4J

Before implementing a keyword, define the keyword as part of a keyword-driven test.

To implement a keyword for reuse in keyword-driven tests:

1. Open a keyword-driven test that includes the keyword that you want to implement.
2. In the **Keyword-Driven Test Editor**, click **Implement Keyword** to the left of the keyword that you want to implement. The **Select Keyword Location** dialog box opens.
3. Click **Select** to select the package and class to which you want to add the keyword implementation.
4. *Optional:* Define the package name for the new keyword implementation in the **Package** field.
5. Define the class name for the new keyword implementation in the **Class** field.
6. Click **OK**.
7. Perform one of the following actions:
 - To record the keyword, click **Yes**.
 - To create an empty keyword method, click **No**.
8. If you have set an application configuration for the current project and you are testing a web application, the **Select Browser** dialog box opens:
 - a) Select the browser.
 - b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
 - c) *Optional:* Select an **Orientation** for the browser window.
 - d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.
9. Click **Record**.
For additional information on recording, see [Recording a Keyword](#).

If an implemented keyword is displayed as not implemented in the **Keywords** window, check **Project > Build Automatically** in the Eclipse menu.

Recording a Keyword in Silk4J

You can only record actions for a keyword that already exists in a keyword-driven test, not for a keyword that is completely new. To record a new keyword-driven test, see [Recording a Keyword-Driven Test](#).

To record the actions for a new keyword:

1. Open a keyword-driven test that includes the keyword that you want to record.
2. In the **Keyword-Driven Test Editor**, click **Implement Keyword** to the left of the keyword that you want to implement. The **Select Keyword Location** dialog box opens.
3. Click **Select** to select the package and class to which you want to add the keyword implementation.
4. *Optional:* Define the package name for the new keyword implementation in the **Package** field.
5. Define the class name for the new keyword implementation in the **Class** field.
6. Click **OK**.
7. If you have set an application configuration for the current project and you are testing a web application, the **Select Browser** dialog box opens:
 - a) Select the browser.
 - b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
 - c) *Optional:* Select an **Orientation** for the browser window.
 - d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.

8. Click **Record**. The **Recording** window opens and Silk4J starts recording the actions for the keyword.
9. In the application under test, perform the actions that you want to test.
For information about the actions available during recording, see *Actions Available During Recording*.
10. Click **Stop**. The **Record Complete** dialog box opens.

The recorded actions are displayed in the context of the defined class.

Marking a Test Method in a Script as a Keyword

Mark an existing test method in a script as a keyword to reuse the method in keyword-driven tests.

1. Open the script which includes the test method that you want to mark as a keyword.
2. Add `@keyword()` to the start of the test method.
By default, the keyword name is the name of the test method.
3. *Optional:* You can set a different name for the keyword by adding `@keyword("keywordName")` to the start of the test method.

You can now use the test method as a keyword in a keyword-driven test.

Examples

To mark the test method `login` as a new keyword with the name *Login*, type the following before the start of the test method:

```
@Keyword("Login")
```

To mark the test method `login` as a new keyword with the name *Login* and with the two input parameters `UserName` and `PassWord`, type the following:

```
@Keyword(value="Login", description="Logs in with the given  
name and password.")  
public void login(@Argument("UserName") String userName,  
@Argument("PassWord") String password) {  
    ... // method implementation  
}
```



Note: If you are viewing this help topic in PDF format, this code sample might include line-breaks which are not allowed in scripts. To use this code sample in a script, remove these line-breaks.

Editing a Keyword-Driven Test



Note: In Silk4J, you can edit and execute keyword-driven tests that are located in Silk4J, and you can execute keyword-driven tests that are stored in Silk Central. To edit a keyword-driven test, which is stored in Silk Central, open the keyword-driven test in the **Keyword-Driven Test Editor** and click **Edit**.

To edit a keyword-driven test:

1. Open the keyword-driven test in the **Keyword-Driven Test Editor**.
 - a) In the **Package Explorer**, expand the project in which the keyword-driven test resides.
 - b) Expand the **Keyword Driven Tests** folder.
 - c) Double-click the keyword-driven test that you want to edit.
2. To add a new keyword to the keyword-driven test:

- a) Click into the **New Keyword** field.
 - b) Type a name for the new keyword.
 - c) Press **Enter**.
3. To edit an existing keyword, click **Open Keyword** to the left of the keyword.
-  **Note:** Silk Central has the ownership of any keyword that has been created in Silk Central, which means any changes that you make to such keywords are saved in Silk Central, not in Silk4J.
4. To copy a keyword into the keyword-driven test:
- a) Select the keyword.
-  **Tip:** Use **Ctrl+Click** or **Shift+Click** on the row number column to select multiple keywords.
- b) Press **Ctrl+C**.
 - c) Select the row above which you want to insert the keyword.
 - d) Press **Ctrl+V**.
5. To move a keyword to another location in the keyword-driven test, click on the keyword and drag it to the new location, or:
- a) Select the keyword.
-  **Tip:** Use **Ctrl+Click** or **Shift+Click** on the row number column to select multiple keywords.
- b) Press **Ctrl+X**.
 - c) Select the row above which you want to insert the keyword.
 - d) Press **Ctrl+V**.
6. To remove the keyword from the keyword-driven test, click **Delete Keyword** to the left of the keyword. The keyword is still available in the **Keywords** window and you can re-add it to the keyword-driven test at any time.
7. To save your changes, click **File > Save**.

Managing Keywords in a Test in Silk Central

The **Keywords** page enables you to manage the keywords of the selected keyword-driven test. The following actions are possible:

Task	Steps
Opening a test or keyword sequence in Silk Test	Click Open with Silk Test to open the selected test or keyword sequence in Silk Test.
Adding a keyword	<ol style="list-style-type: none"> 1. Click New Keyword at the bottom of the keywords list, or right-click a keyword and select Insert Keyword Above from the context menu. <ul style="list-style-type: none">  Note: You can let Silk Test recommend keywords based on their usage. Toggle the recommendations on or off with Enable Recommendations or Disable Recommendations in the context menu. For additional information, see <i>Which Keywords Does Silk4J Recommend?</i> 2. Select a keyword from the list of available keywords or type a new name to create a new keyword. 3. Click Save. <p>Alternatively, double click an existing keyword in the All Keywords pane on the right or drag and drop it.</p>

Task	Steps
	 Tip: You can select multiple keywords with Ctrl+Click . When dropping them, they will be sorted in the order that you selected them in.
Deleting a keyword	Click  in the Actions column of the keyword that you want to delete. Click Save .
Changing the order of keywords	Drag and drop a keyword to the desired position. Click Save .
Creating a keyword sequence (a keyword consisting of other keywords)	<ol style="list-style-type: none"> 1. Select the keywords that you want to combine in the keywords list. Use Ctrl+Click or Shift+Click on the row number column to select multiple keywords. 2. Right-click your selection and click Combine. 3. Specify a Name and Description for the new keyword sequence.
Extracting keywords from a keyword sequence	Right-click a keyword sequence and click Extract keywords . The original keyword sequence is then replaced by the keywords that it contained, but it is not removed from the library. Click Save .
Copying and pasting keywords into tests or keyword sequences	<ol style="list-style-type: none"> 1. Select the keywords that you want to copy in the keywords list. Use Ctrl+Click or Shift+Click on the row number column to select multiple keywords. 2. Press Ctrl+C to copy your selection, or Ctrl+X if you want to move the keywords. 3. Open the test or keyword sequence that you want to copy the keywords to and select the row above which the keywords will be inserted. 4. Press Ctrl+V. <p> Tip: You can also paste your selected keywords into Excel, and copy and paste them from there into your tests or keyword sequences.</p>
Defining parameters for a keyword sequence	<ol style="list-style-type: none"> 1. Click Parameters above the keywords list. The Parameters dialog box appears. 2. Click Add Parameter. 3. Specify a Name for the new parameter. If the parameter is an outgoing parameter (delivers a value, instead of requiring an input value), check the Output checkbox. 4. Click OK. 5. Click Save. <p> Note: A keyword or a keyword sequence can have a combined total of 20 input and output parameters.</p>
Editing a draft keyword	<ol style="list-style-type: none"> 1. Click  in the Actions column of the draft keyword that you want to edit. 2. Select a Group or specify a new group for the keyword. 3. Type a Description for the keyword. This information is valuable for the engineer who will implement the keyword. 4. Click OK. 5. <i>Optional:</i> Click into a parameter field to add parameters for the keyword. If the keyword is implemented with Silk Test, these parameters will appear in the generated code stub. 6. Click Save.

Task	Steps
Searching for a keyword	<p>Use the search field in the Keywords view to find a specific keyword. When you enter alphanumeric characters, the list is dynamically updated with all existing matches. Tips for searching:</p> <ul style="list-style-type: none"> • The search is case-insensitive: <code>doAction</code> will find <code>doaction</code> and <code>DOAction</code>. • Enter only capital letters to perform a so-called <i>Camel/Case</i> search: <code>ECD</code> will find <code>Enter Car Details</code>, <code>Enter Contact Details</code> and <code>EnterContactDetails</code>. • Keyword and group names are considered: <code>test</code> will find all keywords that contain <code>test</code> and all keywords in groups where the group name contains <code>test</code>. • <code>?</code> replaces 0-1 characters: <code>user?test</code> will find <code>userTest</code> and <code>usersTest</code>. • <code>*</code> replaces 0-n characters: <code>my*keyword</code> will find <code>myKeyword</code>, <code>myNewKeyword</code> and <code>my_other_keyword</code>. • <code><string>.</code> only searches in group names: <code>group.</code> will find all keywords in groups where the group name contains <code>group</code>. • <code>.<string></code> only searches in keyword names: <code>.keyword</code> will find all keywords that contain <code>keyword</code>. • <code><string>.<string></code> searches for a keyword in a specific group: <code>group.word</code> will find <code>myKeyword</code> in the group <code>myGroup</code>. • Use quotes to search for an exact match: <code>'Keyword'</code> will find <code>Keyword</code> and <code>MyKeyword</code>, but not <code>keyword</code>.

Which Keywords Does Silk4J Recommend?

When you add keywords to a keyword-driven test or a keyword sequence in the **Keyword-Driven Test Editor**, Silk4J recommends existing keywords which you might want to use as the next keyword in your test. The recommended keywords are listed on top of the keywords list, and are indicated by a bar graph, with the filled-out portion of the graph corresponding to how much Silk4J recommends the keyword.

Silk4J recommends the keywords based on the following:

- When you add the first keyword to a keyword-driven test or a keyword sequence, Silk4J searches for similar keywords that are used as the first keyword in other keyword-driven tests or keyword sequences. The keywords that are used most frequently are recommended higher.
- When you add additional keywords to a keyword-driven test or a keyword sequence, which already includes other keywords, Silk4J recommends keywords as follows:
 - If there are keywords before the position in the keyword-driven test or the keyword sequence, to which you add a new keyword, Silk4J compares the preceding keywords with keyword combinations in all other keyword-driven tests and keyword sequences and recommends the keywords that most frequently follow the preceding combination of keywords.
 - If there are no keywords before the position in the keyword-driven test or the keyword sequence, but there are keywords after the current position, then Silk4J compares the succeeding keywords with keyword combinations in all other keyword-driven tests and keyword sequences and recommends the keywords that most frequently precede the succeeding combination of keywords.
- Additionally, Silk4J takes into account how similar the found keywords are. For example, if both the name and group of two keywords match, then Silk4J recommends these keywords higher in comparison to two keywords for which only the name matches.

- If you have established a connection with Silk Central, any keywords included in keyword-driven tests, which belong to the keyword library that corresponds to the current project, are also considered.

Using Parameters with Keywords

A keyword or a keyword sequence can have a combined total of 20 input and output parameters. This topic describes how you can handle these parameters with Silk4J.

In the **Keyword-Driven Test Editor**, you can view any defined parameters for a keyword or a keyword sequence and you can edit the parameter values.

In the **Keywords** window, you can see which parameters are assigned to a keyword or a keyword sequence when you hover the mouse cursor over the keyword or keyword sequence.

Input parameters for simple keywords

You can define and use input parameters for keywords in the same way as for any other test method.

The following code sample shows how you can define the keyword `setUserDetails` with the two input parameters `userName` and `password`:

```
@Keyword
public void setUserDetails(String userName, String password) {
    ...
}
```

Output parameters for simple keywords

You can define a return value or one or more output parameters for a keyword. You can also use a combination of a return value and one or more output parameters.

The following code sample shows how you can define the keyword `getText` that returns a string:

```
@Keyword
public String getText() {
    return "text";
}
```

The following code sample shows how you can define the keyword `getUserDetails` with the two output parameters `userName` and `password`:

```
@Keyword
public void getUserDetails(OutParameter userName, OutParameter password) {
    userName.setValue("name");
    password.setValue("password");
}
```

Parameters for keyword sequences

You can define or edit the parameters for a keyword sequence in the **Parameters** dialog box, which you can open if you click **Parameters** in the **Keyword Sequence Editor**.

Example: Keywords with Parameters

This topic provides an example of how you can use keywords with parameters. A keyword or a keyword sequence can have a combined total of 20 input and output parameters.

As a first step, create a keyword-driven test which contains the keywords that you want to use. You can do this by recording an entire keyword-driven test, or by creating a new keyword-driven test and by adding the keywords in the keyword-driven test editor.

In this example, the keyword-driven test includes the following keywords:

- Start application** This is the standard keyword that starts the AUT and sets the base state.
- Login** This keyword logs into the AUT with a specific user, identified by a user name and a password.
- GetCurrentUser** This keyword returns the name of the user that is currently logged in to the AUT.
- AssertEquals** This keyword compares two values.
- Logout** This keyword logs the user out from the AUT.

The next step is to add the parameters to the keywords. To do this, open the test scripts of the keywords and add the parameters to the methods.

To add the input parameters `UserName` and `Password` to the keyword `Login`, change

```
@Keyword("Login")
public void login() {
    ...
}
```

to

```
@Keyword("Login")
public void login(String UserName, String Password) {
    ...
}
```

To add the output parameter `UserName` to the keyword `GetCurrentUser`, change

```
@Keyword("GetCurrentUser")
public void getCurrentUser() {
    ...
}
```

to

```
@Keyword("GetCurrentUser")
public void getCurrentUser(OutParameter CurrentUser) {
    ...
}
```

The keyword-driven test in the **Keyword-Driven Test Editor** should look similar to the following:

		Keyword	Parameters	
1	 	Start application		
2	 	Login	<i>UserName</i> <i>Password</i>	
3	 	GetCurrentUser	<i>CurrentUser ←</i>	
4	 	AssertEquals	<i>Expected</i> <i>Actual</i>	
5	 	Logout		

Now you can specify actual values for the input parameters in the **Keyword-Driven Test Editor**. To retrieve the value of the output parameter `UserName` of the keyword `GetCurrentUser`, provide a variable, for example ``${current user}`. You can then pass the value that is stored in the variable to subsequent keywords.

		Keyword	Parameters	
1	 	Start application		
2	 	Login	UserName	Password
3	 	GetCurrentUser	\${current user}	
4	 	AssertEquals	John Smith	\${current user}
5	 	Logout		

Combining Keywords into Keyword Sequences

Use the **Keyword-Driven Test Editor** to combine keywords, which you want to execute sequentially in multiple keyword-driven tests, into a keyword sequence.

1. Open the keyword-driven test that includes the keywords that you want to combine.
2. In the **Keyword-Driven Test Editor**, press and hold down the **Ctrl** key and then click the keywords that you want to combine.
3. Right-click on the selection and click **Combine**. The **Combine Keywords** dialog box opens.
4. Type a name for the new keyword sequence into the **Name** field.
5. *Optional:* Type a description for the new keyword sequence into the **Description** field.
6. Click **Combine**.

The new keyword sequence opens and is also displayed in the **Keywords** window. You can use the keyword sequence in keyword-driven tests.



Note: Like any other keyword, you cannot execute a keyword sequence on its own, but only as part of a keyword-driven test.

Replaying Keyword-Driven Tests from Eclipse

1. In the **Project Explorer**, navigate to the keyword-driven test asset that you want to replay.
2. Right-click the asset name.
3. Choose **Run As > Keyword-Driven Test**.
4. *Optional:* To open the **Run Configurations** dialog box, choose **Run As > Run Configurations**.
5. *Optional:* In the **Run Configurations** dialog box, you can select a different test or project.
6. *Optional:* In the **Global variables** grid of the **Run Configurations** dialog box, you can set the values of any variables that are used for the execution of the keyword-driven test. These values are used whenever you execute the keyword-driven test asset.
 - a) Type a **Variable Name** and a **Value** for the variable into the corresponding fields.
 - b) Type **Enter** to add a new line to the grid.
 - c) Repeat the previous two steps until you have set the values of all the global variables that you want to use.

When executing keyword-driven tests that are part of an automation framework and that are managed in a test management tool, for example Silk Central, you can also add a new *.properties* file to a project to set the values of global variables for the entire project. For additional information, see *Replaying a Keyword-Driven Test with Specific Variables*.

7. *Optional:* To close the **Run Configurations** dialog box and to start the execution of the keyword-driven test asset, click **Run**.

8. If you are testing a web application, the **Select Browser** dialog box opens. Select the browser and click **Run**.
9. Click **Run**.
10. *Optional:* If necessary, you can press both **Shift** keys at the same time to stop the execution of the test.
11. When the test execution is complete, the **Playback Complete** dialog box opens. Click **Explore Results** to review the TrueLog for the completed test.

Replaying Keyword-Driven Tests Which Are Stored in Silk Central

Replay a keyword-driven test that is stored in Silk Central to verify that the tested functionality is behaving as expected.

1. In the Silk4J menu, click **Silk4J > Show Keywords View**.
2. In the **Keywords** view, double-click the keyword-driven test.
To update the **Keywords** view with any changes from Silk Central, click **Refresh**.
3. In the toolbar, click **Run**.
4. If you are testing a web application, the **Select Browser** dialog box opens. Select the browser and click **Run**.
5. Click **Run**.
6. *Optional:* If necessary, you can press both **Shift** keys at the same time to stop the execution of the test.
7. When the test execution is complete, the **Playback Complete** dialog box opens. Click **Explore Results** to review the TrueLog for the completed test.

Replaying Keyword-Driven Tests from the Command Line

You must update the PATH variable to reference your JDK location before performing this task. For additional information, see [JDK Installation for Microsoft Windows](#).

To replay keyword-driven tests from the command line, for example when replaying the tests from a CI server, use the `KeywordTestSuite` class.

1. To execute a keyword-driven test from the command line, create a JUnit test suite with the `@KeywordTests` annotation. For example, if you want to execute the keyword-driven test *My Keyword-Driven Test*, create the JUnit test suite `MyTestSuite` as follows:

```
@RunWith(KeywordTestSuite.class)
@KeywordTests({ "My Keyword-Driven Test" })
public class MyTestSuite {
}

```

2. Include the following in the CLASSPATH:

- `junit.jar`.
- The `org.hamcrest.core` JAR file.
- `silktest-jtf-nodeps.jar`.
- `com.borland.silk.keyworddriven.engine.jar`.
- The JAR of folder that contains your keyword-driven tests.

```
set CLASSPATH=<eclipse_install_directory>\plugins
\org.junit_4.11.0.v201303080030\junit.jar;<eclipse_install_directory>
\plugins\org.hamcrest.core_1.3.0.v201303031735.jar;%OPEN_AGENT_HOME%\JTF

```

```
\silktest-jtf-nodeps.jar;%OPEN_AGENT_HOME%\KeywordDrivenTesting  
\com.borland.silk.keyworddriven.engine.jar;C:\myTests.jar
```

3. *Optional:* Add a new `.properties` file to the project to set the values of any variables that are used for the execution of the keyword-driven test.

For additional information, see *Replaying a Keyword-Driven Test with Specific Variables*.

4. Run the JUnit test method by typing `java org.junit.runner.JUnitCore <Name>`, where the *Name* is the name of the JUnit test suite that you have created in the first step.



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

Example

For example, to run the two keyword driven tests *My Keyword Driven Test 1* and *My Keyword Driven Test 2*, create the following class:

```
package demo;  
  
import org.junit.runner.RunWith;  
  
import com.borland.silktest.jtf.keyworddriven.KeywordTestSuite;  
import com.borland.silktest.jtf.keyworddriven.KeywordTests;  
  
@RunWith(KeywordTestSuite.class)  
@KeywordTests({ "My Keyword Driven Test 1", "My Keyword Driven  
Test 2" })  
public class MyTestSuite {  
  
}
```

To run the class from the command line, type the following:

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

To run the class from the command line, using global variables stored in the file `c:\temp\globalvariables.properties`, type the following:

```
java -Dsilk.keyworddriven.engine.globalVariablesFile=c:\temp  
\globalvariables.properties org.junit.runner.JUnitCore  
demo.MyTestSuite
```

For additional information, see *Replaying a Keyword-Driven Test with Specific Variables*.

Replaying Keyword-Driven Tests with Apache Ant

To perform the actions described in this topic, ensure that Apache Ant is installed on your machine.

To replay keyword-driven tests with Apache Ant, for example to generate HTML reports of the test runs, use the `KeywordTestSuite` class.

1. To execute a keyword-driven test with Apache Ant, create a JUnit test suite with the `@KeywordTests` annotation. For example, if you want to execute the keyword-driven test *My Keyword-Driven Test*, create the JUnit test suite `MyTestSuite` as follows:

```
@RunWith(KeywordTestSuite.class)  
@KeywordTests({ "My Keyword-Driven Test" })  
public class MyTestSuite {  
  
}
```

2. Open the `build.xml` file of the Silk4J project, which includes the keyword-driven test.

3. To execute the keyword-driven test, add the following target to the `build.xml` file:

```
<target name="runTests" depends="compile">
  <mkdir dir="./reports"/>
  <junit printsummary="true" showoutput="true" fork="true">
    <classpath>
      <fileset dir="${output}">
        <include name="**/*.jar" />
      </fileset>
      <fileset dir="${buildlib}">
        <include name="**/*.jar" />
      </fileset>
      <fileset dir="C:/Program Files (x86)/Silk/SilkTest/ng/
KeywordDrivenTesting">
        <include name="**/*.jar" />
      </fileset>
    </classpath>

    <test name="MyTestSuite" todir="./reports"/>
  </junit>
</target>
```

For additional information about the `JUnit` task, see <https://ant.apache.org/manual/Tasks/junit.html>.

4. *Optional:* To create XML reports for all tests, add the following code to the target:

```
<formatter type="xml" />
```

5. *Optional:* To create HTML reports out of the XML reports, add the following code to the target:

```
<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml" />
  </fileset>
  <report format="noframes" todir="./report/html" />
</junitreport>
```

For additional information about the `JUnitReport` task, see <https://ant.apache.org/manual/Tasks/junitreport.html>.

The complete target should now look like the following:

```
<target name="runTests" depends="compile">
  <mkdir dir="./reports"/>
  <junit printsummary="true" showoutput="true" fork="true">
    <classpath>
      <fileset dir="${output}">
        <include name="**/*.jar" />
      </fileset>
      <fileset dir="${buildlib}">
        <include name="**/*.jar" />
      </fileset>
      <fileset dir="C:/Program Files (x86)/Silk/SilkTest/ng/
KeywordDrivenTesting">
        <include name="**/*.jar" />
      </fileset>
    </classpath>

    <test name="MyTestSuite" todir="./reports"/>
  </junit>
  <junitreport todir="./reports">
    <fileset dir="./reports">
      <include name="TEST-*.xml" />
    </fileset>
    <report format="noframes" todir="./report/html" />
  </junitreport>
</target>
```

```
</junitreport>  
</target>
```

6. To run the tests from Eclipse, perform the following actions:
 - a) In the **Package Explorer**, right-click the `build.xml` file.
 - b) Select **Run As > Ant Build ...**
 - c) In the **Targets** tab of the **Edit Configuration** dialog box, check **runTests**.
 - d) Click **Run**.

You can also execute the tests from the command line or from a CI server. For additional information, see <https://ant.apache.org/manual/running.html> and *Replaying Tests from a Continuous Integration Server* in the *Silk4J Help*.

Replaying a Keyword-Driven Test with Specific Variables

Before you can set the values of variables for the execution of a keyword-driven test, you have to create the project.

To set the values of global variables for all executions of a keyword-driven test asset, where these executions are triggered by you, use the **Global variables** grid of the **Run Configurations** dialog box. For additional information, see *Replaying Keyword-Driven Tests from Eclipse*.

When executing keyword-driven tests that are part of an automation framework and that are managed in a test management tool, for example Silk Central, you can set the values of any variables that are used for the execution of the keyword-driven test in Silk4J. To set the values of global variables for the entire project, which means that these values are used whenever a Silk4J user executes the keyword-driven test assets in this project, perform the following actions:

1. In the **Package Explorer**, expand the project which includes the keyword-driven tests that you want to execute based on the variables.
2. Right-click the folder **src** of the project and select **New > File**. The **New File** dialog box opens.
3. Type `globalvariables.properties` into the **File name** field.
4. Click **Finish**. The new properties file opens.
5. Add new lines to the file to specify the variables.

The format for a new variable is:

```
name=value
```

For example, to specify the two variables `user` and `password`, type the following:

```
user=John  
password=john5673
```

For information about the format of a properties file and how you can enter UNICODE characters, for example a space, see [Properties File Format](#).

6. Save the `globalvariables.properties` file.
7. Open the keyword-driven test that you want to execute.
8. In the **Keyword-Driven Test Editor**, edit the parameters to use the new variables.

Use the following annotation:

```
${variable name}
```

For example, in the following keyword-driven test, the `current user` parameter uses a global variable:

		Keyword	Parameters	
1	 	Start application		
2	 	Login	UserName	Password
3	 	GetCurrentUser	\${current user}	
4	 	AssertEquals	John Smith	\${current user}
5	 	Logout		

Whenever a keyword-driven test in the project is executed from Silk4J, the variables are used.

Integrating Silk4J with Silk Central

Integrate Silk4J and Silk Central to enable collaboration between technical and less-technical users.

When Silk4J and Silk Central are integrated and a library with the same name as the active Silk4J project exists in Silk Central, the **Keywords** view under **Silk4J > Show Keywords View** displays all keywords from the Silk Central library in addition to any keywords defined in the active Silk4J project.



Note: The Silk Central connection information is separately stored for every Silk4J user, which means every Silk4J user that wants to work with keywords and keyword sequences from Silk Central must integrate Silk4J with Silk Central.

Integrating Silk4J with Silk Central provides you with the following advantages:

- Test management and execution is handled by Silk Central.
- Keywords are stored in the Silk Central database (upload library) and are available to all projects in Silk Central.
- Manual tests can be directly automated in Silk Central and the created keyword-driven tests can be executed in Silk4J from Silk Central.



Note: In Silk4J, you can edit and execute keyword-driven tests that are located in Silk4J, and you can execute keyword-driven tests that are stored in Silk Central. To edit a keyword-driven test, which is stored in Silk Central, open the keyword-driven test in the **Keyword-Driven Test Editor** and click **Edit**.

1. From the Eclipse menu, select **Silk4J > Silk Central Configuration**. The **Preferences** dialog box opens.

2. Type the URL of your Silk Central server into the **URL** field.

For example, if the Silk Central server name is *sctm-server*, and the port for Silk Central is 13450, type `http://sctm-server:13450`.

3. Specify the web-service token for authentication.

You can generate a web-service token in the **User Settings** page of Silk Central, which you can access by clicking on the user name in the Silk Central menu.



Note: To authenticate with your Silk Central user name and password, you could select **User name and password** from the **Authentication** list. However, for security reasons, Micro Focus recommends using a web-service token for authentication instead of sending your user name and password over the network.

4. Click **Verify** to verify if Silk4J can access the Silk Central server with the specified user.

5. Click **OK**.

Implementing Silk Central Keywords in Silk4J

Before implementing Silk Central keywords, define the keywords as part of a keyword-driven test in Silk Central.

To implement a Silk Central keyword in Silk4J:

1. Create a project in Silk4J with the same name as the keyword library in Silk Central, which includes the keyword-driven test.
2. If the keyword library in Silk Central has no type assigned, click **Silk4J > Upload Keyword Library** to set the library type.
3. *Optional:* To implement a specific keyword in Silk4J from Silk Central, open the **Keywords** tab of the library in Silk Central and click **Implement with Silk Test** in the **Actions** column of the keyword.
4. In the Silk4J menu, click **Silk4J > Show Keywords View**.
5. In the **Keywords** view, double-click the keyword-driven test.
To update the **Keywords** view with any changes from Silk Central, click **Refresh**.
6. In the toolbar, click **Record Actions**.
7. If you have set an application configuration for the current project and you are testing a web application, the **Select Browser** dialog box opens:
 - a) Select the browser.
 - b) *Optional:* If you want to test a web application on a desktop browser with a predefined browser size, select the browser size from the **Browser size** list.
For example, to test a web application on Apple Safari and in a browser window which is as big as the screen of the Apple iPhone 7, select **Apple iPhone 7** from the list.
 - c) *Optional:* Select an **Orientation** for the browser window.
 - d) *Optional:* Click **Edit Browser Sizes** to specify a new browser size and to select which browser sizes should be shown in the **Browser size** list.
8. Click **Record**.
For additional information on recording, see [Recording a Keyword](#).
9. Record the actions for the first unimplemented keyword.
10. When you have recorded all the actions for the current keyword, click **Next Keyword**.
11. To switch between keywords in the **Recording** window, click **Previous Keyword** and **Next Keyword**.
12. Click **Stop**. The **Record Complete** dialog box opens.



Note: You cannot delete keywords or change the sequence of the keywords in a keyword-driven test from Silk Central, as these tests are read only in Silk4J.

If an implemented keyword is displayed as not implemented in the **Keywords** window, check **Project > Build Automatically** in the Eclipse menu.

Uploading a Keyword Library to Silk Central

To work with Silk Central, ensure that you have configured a valid Silk Central location. For additional information, see [Integrating Silk4J with Silk Central](#).

To automate manual tests in Silk Central, upload keywords that you have implemented in a Silk4J project as a keyword library to Silk Central, where you can then use the keywords to automate manual tests.

1. In Silk4J, select the project in which the keyword-driven tests reside.
2. Ensure that a library with the same name exists in Silk Central (**Tests > Libraries**).

3. In the toolbar, click **Upload Keyword Library**.
4. *Optional:* Provide a description of the changes to the keyword library.
5. *Optional:* Click **Configure** to configure the connection to Silk Central.
6. *Optional:* To see which libraries are available in the connected Silk Central instance, click on the link.
7. Click **Upload**.



Caution: If the keyword library in Silk Central is already assigned to a different automation tool or another Silk Test client, you are asked if you really want to change the type of the keyword library. Upload the library only if you are sure that you want to change the type.

Silk4J creates a keyword library out of all the keywords that are implemented in the project. Then Silk4J saves the keyword library with the name `library.zip` into the output folder of the project. The library is validated for consistency, and any changes which might break existing tests in Silk Central are listed in the **Upload Keyword Library to Silk Central** dialog box. Finally, Silk4J uploads the library to Silk Central. You can now use the keywords in Silk Central. Any keyword-driven tests in Silk Central, which use the keywords that are included in the keyword library, automatically use the current implementation of the keywords.

Uploading a keyword library from a project that was created in Silk Test 15.5

To upload keyword libraries from Silk4J projects that were created with Silk Test 15.5, you need to edit the `build.xml` file of the project.

1. In the **Package Explorer**, expand the folder of the project from which you want to upload the keyword library.
2. Open the `build.xml` file.
3. Add the keyword assets directory of the project to the JAR build step of the *compile* target:

```
<fileset dir="Keyword Assets" includes="**/*.kwd"
erroronmissingdir="false" />
```

4. Add the following target for the keyword library:

```
<target name="build.keyword.library" depends="compile">
  <java
    classname="com.borland.silk.kwd.library.docbuilder.DocBuilder"
    fork="true">
    <classpath refid="project.classpath" />

    <arg value="AutoQuote Silk4J Library" />
    <arg value="\${output}" />
    <arg value="\${output}/library.zip" />
  </java>
</target>
```

The new `build.xml` file should look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="AutoQuote" default="compile">

  <property name="src" value="src" />
  <property name="bin" value="build" />
  <property name="output" value="output" />
  <property name="lib" value="lib" />
  <property name="buildlib" value="buildlib" />

  <path id="project.classpath">
    <fileset dir="\${lib}" includes="*.jar"
excludes="*source*" />
    <fileset dir="\${buildlib}" includes="*.jar"
excludes="*source*" />
  </path>
```

```

<target name="clean">
  <delete dir="${output}" />
</target>

<target name="compile" depends="clean">
  <mkdir dir="${output}" />

  <delete dir="${bin}" />
  <mkdir dir="${bin}" />

  <componentdef name="ecj"
class="org.eclipse.jdt.core.JDTCompilerAdapter"
classpathref="project.classpath" />
  <javac srcdir="${src}" destdir="${bin}" debug="true"
source="1.7" target="1.7" encoding="utf-8"
includeantruntime="false">
    <classpath refid="project.classpath" />
    <ecj />
  </javac>

  <jar destfile="${output}/tests.jar" >
    <fileset dir="${bin}" includes="**/*.class" />
    <fileset dir="${src}" includes="**/*" excludes="**/*
*.java" />
    <fileset dir="Object Maps" includes="**/*.objectmap"
erroronmissingdir="false" />
    <fileset dir="Image Assets" includes="**/*.imageasset"
erroronmissingdir="false" />
    <fileset dir="Verifications" includes="**/*.verification"
erroronmissingdir="false" />
    <fileset dir="Keyword Assets" includes="**/*.kwd"
erroronmissingdir="false" />
  </jar>

  <copy todir="${output}" overwrite="true">
    <fileset dir="${lib}" includes="*.jar"
excludes="*source*" />
  </copy>
  <delete dir="${bin}" />
</target>

<target name="build.keyword.library" depends="compile">
  <java
class="com.borland.silk.kwd.library.docbuilder.DocBuilder"
fork="true">
    <classpath refid="project.classpath" />

    <arg value="AutoQuote Silk4J Library" />
    <arg value="${output}" />
    <arg value="${output}/library.zip" />
  </java>
</target>
</project>

```

Uploading a Keyword Library to Silk Central from the Command Line

Upload an external keyword library to Silk Central from a Java-based command line to integrate Silk Central and your keyword-driven tests into your continuous integration build system, for example Jenkins.

To upload your keyword library to Silk Central from a Java-based command line:

1. Select **Help > Tools** in Silk Central and download the **Java Keyword Library Tool**.
2. Call the command line tool that is contained in the downloaded `jar` file with the following arguments:

- `java`
- `-jar com.borland.silk.keyworddriven.jar`
- `-upload`
- `Library` name of the library in Silk Central to be updated, or created if it does not yet exist.
- `Package` name of the library package (`zip` archive) to be uploaded.
- `Hostname:port` of the Silk Central front-end server.
- `Web-service` token of the Silk Central user. Required for authentication. You can generate a web-service token in the **User Settings** page of Silk Central, which you can access by clicking on the user name in the Silk Central menu.



Note: For security reasons, Micro Focus recommends using a web-service token for authentication instead of sending your user name and password over the network.

- `Username` of the Silk Central user. Not required when using a web-service token for authentication.
- `Password` of the Silk Central user. Not required when using a web-service token for authentication.
- `Update information`, describing the changes that were applied to the library, in quotes.
- `[-allowUsedKeywordDeletion]`, an optional flag to allow the deletion of keywords that are used in a test or keyword sequence. By default, an error is raised if used keywords are attempted to be deleted.

The following example outlines the command line to upload a library to Silk Central with Java 9 or later:

```
java --add-modules=java.activation,java.xml.ws -jar
com.borland.silk.keyworddriven.jar -upload
"My library" "./output/library.zip" silkcentral:19120 scLogin
scPassword "Build xy: Implemented missing keywords"
```

Examples

The following example outlines the command line to upload a library to Silk Central with Java 8 or prior by using a web-service token for authentication:

```
java -jar com.borland.silk.keyworddriven.jar -upload
"My library" "./output/library.zip" silkcentral:19120 scToken
"Build xy: Implemented missing keywords"
```

To upload the same library with Java 8 or prior by using user name and password for authentication, use a command like the following:

```
java -jar com.borland.silk.keyworddriven.jar -upload
"My library" "./output/library.zip" silkcentral:19120 scLogin
scPassword "Build xy: Implemented missing keywords"
```

The corresponding commands with Java 9 or later are:

```
java --add-modules=java.activation,java.xml.ws -jar
com.borland.silk.keyworddriven.jar -upload
"My library" "./output/library.zip" silkcentral:19120 scToken
"Build xy: Implemented missing keywords"
```

```
java --add-modules=java.activation,java.xml.ws -jar
com.borland.silk.keyworddriven.jar -upload
"My library" "./output/library.zip" silkcentral:19120 scLogin
scPassword "Build xy: Implemented missing keywords"
```



Note: When uploading a keyword-driven library with Java 9 or later, ensure `JAVA_HOME` is defined on the execution servers and points to a JDK with the corresponding Java version.

Searching for a Keyword

Use the search field in the **Keywords** view to find a specific keyword. When you enter alphanumeric characters, the list is dynamically updated with all existing matches. Tips for searching:

- The search is case-insensitive: `doAction` will find `doaction` and `DOAction`.
- Enter only capital letters to perform a so-called *Camel/Case* search: `ECD` will find `Enter Car Details`, `Enter Contact Details` and `EnterContactDetails`.
- Keyword and group names are considered: `test` will find all keywords that contain `test` and all keywords in groups where the group name contains `test`.
- `?` replaces 0-1 characters: `user?test` will find `userTest` and `usersTest`.
- `*` replaces 0-n characters: `my*keyword` will find `myKeyword`, `myNewKeyword` and `my_other_keyword`.
- `<string>.` only searches in group names: `group.` will find all keywords in groups where the group name contains `group`.
- `.<string>` only searches in keyword names: `.keyword` will find all keywords that contain `keyword`.
- `<string>.<string>` searches for a keyword in a specific group: `group.word` will find `myKeyword` in the group `myGroup`.
- Use quotes to search for an exact match: `'Keyword'` will find `Keyword` and `MyKeyword`, but not `keyword`.

Filtering Keywords

To find a specific keyword in the current project, you can filter the keywords that are displayed in the **Keywords** window. If an integration with Silk Central is configured, the result includes the relevant keywords from Silk Central.

1. In the menu, click **Silk4J > Show Keywords View** to open the **Keywords** window.
2. In the **Keywords** window, type the name of the keyword that you are searching for into the search field. The **Keywords** window lists all keywords in the current project with the given name.
3. *Optional:* To see in which keyword-driven tests and keyword sequences a keyword is used, hover the mouse cursor over the keyword in the **Keywords** window and click **Find Keyword Usages**.
If an integration with Silk Central is configured, the result includes the relevant keywords from Silk Central.
4. *Optional:* To edit a keyword, hover the mouse cursor over the keyword in the **Keywords** window and click **Go to implementation**.

Finding All References of a Keyword

To find all keyword-driven tests and Java files in which a keyword is referenced:

1. In the **Keyword-Driven Test Editor**, click **Open Keyword**. The Java file, in which the keyword is implemented, opens.
2. Right-click on the name of the method that implements the keyword.
3. Click **References**.
4. To find all references of the keyword in the workspace, click **Workspace**.

All keyword-driven tests and Java files in which the keyword is referenced are listed in the **Search** window.

Grouping Keywords

To better structure the keywords in a library, you can group them.

This topic shows how you can add a keyword to a specific group. These group names are also used by Silk Central and your keywords are grouped accordingly.

To add a keyword to a specific group:

1. Open the implementation of the keyword.
 - a) Open the project in which the keyword is implemented.
 - b) Open the **Keywords** window.
 - c) In the **Keywords** window, select the keyword.
 - d) Click **Go to implementation**.
2. To add all methods in a class to the keyword group, add the keyword group before the start of the class. For example, to add the group calculator to the keywords, type:

```
@KeywordGroup("Calculator")
```

In the **Keywords** window, the displayed keyword name now includes the group. For example, the keyword *Addition* in the group *Calculator* is displayed as *Calculator.Addition*.

Troubleshooting for Keyword-Driven Testing

Why does the Keywords window falsely show a keyword as not implemented?

If an implemented keyword is displayed as not implemented in the **Keywords** window, check **Project > Build Automatically** in the Eclipse menu.

Why do I get the error "No application configuration present" when trying to replay a keyword-driven test from Silk Central?

If you get this error, your keyword-driven test does not include a *Start application* keyword as the first keyword. Silk4J requires the *Start application* keyword to apply the application configuration of your project to the keyword-driven test. When you record a new keyword-driven test, Silk4J automatically adds the *Start application* keyword as the first keyword to the keyword-driven test.

With Silk4J 19.5 or later, parallel testing is enabled by default. When you upgrade from a Silk4J version prior to Silk4J 19.5 to a more recent Silk4J version, previously executing keyword-driven tests might no longer execute because each individual test is now expected to specify the application under test (AUT).

To work around this issue, disable parallel testing on the machine on which Silk4J is running by setting the environment variable `SILKTEST_ENABLE_PARALLEL_TESTING` to false.

How can I prevent my browser or mobile app from closing between tests?

When replaying multiple keyword-driven tests from Silk Central with Silk4J 19.5 or later, parallel testing is enabled by default. When you upgrade from a Silk4J version prior to Silk4J 19.5 to a more recent Silk4J version, Silk4J will by default close the browser or the mobile app whenever a keyword-driven test is finished.

This issue does not occur when testing a browser application against Internet Explorer. To work around this issue, disable parallel testing on the machine on which Silk4J is running by setting the environment variable `SILKTEST_ENABLE_PARALLEL_TESTING` to false.

Object Recognition

Silk4J identifies any control in the application under test (AUT) by combining the name of the control class and a collection of prioritized attributes into a unique XPath locator. If the combined XPath locator does not uniquely identify the control, Silk4J additionally adds an index to the locator. During recording, Silk4J allows you to select an alternative locator for a control from the list in the **Locator** field of the **Choose Action** dialog.

If you are recording WebDriver locators instead of Silk Test locators, Silk4J provides the following alternatives to the XPath locator when identifying a control:

- Locate by id. Identifies the control by the id attribute.
- Locate by name. Identifies the control by the name attribute.
- Locate by link text. Only for hyperlinks.

Within Silk4J, literal references to identified objects are referred to as *locators*. Silk4J uses locators to find and identify objects in the application under test (AUT). Locators are a subset of the XPath query language, which is a common XML-based language defined by the World Wide Web Consortium, W3C.

Locator Basic Concepts

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

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 starting from the current context.
- `Shell` finds all shells that are direct children of the current context.

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

Dynamic object recognition uses a `Find` or `FindAll` functions to identify an object in a test case.

Object Type and Search Scope

A locator typically contains the type of object to identify and a search scope. The search scope is one of the following:

- `//`
- `/`

Locators rely on the current object, which is the object for which the locator is specified. The current object is located in the object hierarchy of the application's UI. All locators depend on the position of the current object in this hierarchy, much like a file system.

XPath expressions rely on the *current context*, which is the position of the object in the hierarchy on which the `Find` method was invoked. All XPath expressions depend on this position, much like a file system.



Note:

The object type in a locator for an HTML element is either the HTML tag name or the class name that Silk4J uses for this object. For example, the locators `//a` and `//DomLink`, where `DomLink` is the

name for hyperlinks in Silk4J, are equivalent. For all non-HTML based technologies only the Silk4J class name can be used.

Example

- `//a` identifies hyperlink objects in any hierarchy relative to the current object.
- `/a` identifies hyperlink objects that are direct children of the current object.



Note: `<a>` is the HTML tag for hyperlinks on a Web page.

Example

The following code sample identifies the first hyperlink in a browser. This example assumes that a variable with the name `browserWindow` exists in the script that refers to a running browser instance. Here the type is "a" and the current object is `browserWindow`.

```
DomLink link = browserWindow.<DomLink>find("//a");
```

Using Attributes to Identify an Object

To identify an object based on its properties, you can use locator attributes. The locator attributes are specified in square brackets after the type of the object.

Example

The following sample uses the `textContent` attribute to identify a hyperlink with the text `Home`. If there are multiple hyperlinks with the same text, the locator identifies the first one.

```
DomLink link = browserWindow.<DomLink>find(//a[@textContent='Home']");
```

Locator Syntax

Silk4J supports a subset of the XPath query language to locate UI controls.

The following table lists the constructs that Silk4J supports.



Note: `<a>` is the HTML tag for hyperlinks on a Web page.

Supported Locator Construct	Sample	Description
<code>//</code>	<code>//a</code>	Identifies objects that are descendants of the current object. The example identifies hyperlinks on a web page.
<code>/</code>	<code>/a</code>	Identifies objects that are direct children of the current object. Objects located on lower hierarchy levels are not recognized.

Supported Locator Construct	Sample	Description
Attribute	<p>Example 1: // a[@textContents='Home']</p> <p>Example 2: // div[@textContents='Price: * USD']</p>	<p>The example identifies hyperlinks on a web page that are direct children of the current object.</p> <p>Identifies objects by a specific attribute. You can use the wildcards * and ? in the attribute value.</p> <p>Example 1 identifies hyperlinks with the text <i>Home</i>, Example 2 uses a wildcard to identify a div with a price.</p>
Index	<p>Example 1: //a[3]</p> <p>Example 2: // a[@textContents='Home'] [2]</p>	<p>Identifies a specific occurrence of an object if there are multiple ones. Indices are 1-based in locators.</p> <p>Example 1 identifies the third hyperlink and Example 2 identifies the second hyperlink with the text <i>Home</i>.</p>
Logical Operators: <ul style="list-style-type: none"> • and • or • not • = • != 	<p>Example 1: // a[@textContents='Remove' or @textContents='Delete']</p> <p>Example 2: // a[@textContents! ='Remove']</p> <p>Example 3: // a[not(@textContents='Delete' or @id='lnkDelete') and @href='*/delete']</p>	<p>Identifies objects by using logical operators to combine attributes.</p> <p>Example 1 identifies hyperlinks that either have the caption <i>Remove</i> or <i>Delete</i>, Example 2 identifies hyperlinks with a text that is not <i>Remove</i>, and Example 3 shows how to combine different logical operators.</p>
ancestor	<p>Example 1: // input[@id='username']/ ancestor::form</p> <p>Example 2: // input[@id='username']/ ancestor::div[@className= 'container']</p>	<p>Identifies ancestors, for example parent, grandparent, and so on, of an object.</p> <p>Example 1 finds the form element that has a child input element with the identifier <i>username</i>, Example 2 finds the div with the class name <i>container</i> that has a child input element with the identifier <i>username</i>.</p>
..	<p>Example 1: // input[@id='username']/ ancestor::form</p> <p>Example 2: // input[@id='username']/ ancestor::div[@className= 'container']</p>	<p>Identifies the parent of an object.</p> <p>Example 1 identifies the parent of the hyperlink with the text <i>Edit</i> and Example 2 identifies a hyperlink with the text <i>Delete</i> that has a sibling hyperlink with the text <i>Edit</i>.</p>

Supported Locator Construct	Sample	Description
following-sibling	Example: // td[@textContents='John']/ following-sibling::td[2]	Identifies siblings after the current object. The example identifies the table cell which is located two cells to the right of the table cell with the text <i>John</i> .
preceding-sibling	Example: // td[@textContents='John']/ preceding-sibling::td[2]	Identifies siblings before the current object. The example identifies the table cell which is located two cells to the left of the table cell with the text <i>John</i> .
*	Example 1: // *[@textContents='Home'] Example 2: /*/a	Identifies objects without considering their types, like hyperlink, text field, or button. Example 1 identifies objects with the given text content, regardless of their type, and Example 2 identifies hyperlinks that are second-level descendants of the current object.

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

Unsupported Locator Construct	Example
Comparing two attributes with each other.	//a[@textContents = @id]
An attribute name on the right side is not supported. An attribute name must be on the left side.	//a['abc' = @id]
Combining multiple locators with <code>and</code> or <code>or</code> .	//a[@id = 'abc'] or ../Checkbox
More than one set of attribute brackets.	//a[@id = 'abc'] [@textContents = '123'] (use //a [@id = 'abc' and @textContents = '123'] instead)
More than one set of index brackets.	//a[1][2]
Any construct that does not explicitly specify a class or the class wildcard, such as including a wildcard as part of a class name.	//[@id = 'abc'] (use //*[@id = 'abc'] instead) "//*/a[@id='abc']"

Using Locators

Within Silk4J, literal references to identified objects are referred to as *locators*. For convenience, you can use shortened forms for the locator strings in scripts. Silk4J automatically expands the syntax to use full locator strings when you playback a script. When you manually code a script, you can omit the following parts in the following order:

- The search scope, `//`.
- The object type name. Silk4J defaults to the class name.

- The surrounding square brackets of the attributes, [].

When you manually code a script, we recommend that you use the shortest form available.



Note: When you identify an object, the full locator string is captured by default.

The following locators are equivalent:

- The first example uses the full locator string.

```
_desktop.<DomLink>find("//BrowserApplication//BrowserWindow//a[@textContents='Home']").select();
```

To confirm the full locator string, use the **Locator Spy** dialog box.

- The second example works when the browser window already exists.

```
browserWindow.<DomLink>find("//a[@textContents='Home']").select();
```

To find an object that has no real attributes for identification, use the index. For instance, to select the second hyperlink on a Web page, you can type:

```
browserWindow.<DomLink>find("//DomLink[2]").select();
```

Additionally, to find the first object of its kind, which might be useful if the object has no real attributes, you can type:

```
browserWindow.<DomLink>find("//DomLink").select();
```

Using Locators to Check if an Object Exists

You can use the `Exists` method to determine if an object exists in the application under test.

The following code checks if a hyperlink with the text *Log out* exists on a Web page:

```
if (browserWindow.exists( "//a[@textContents='Log out']" )) {
    // do something
}
```

Using the Find method

You can use the `Find` method and the `FindOptions` method to check if an object, which you want to use later, exists.

The following code searches for a window and closes the window if the window is found:

```
Window mainWindow = _desktop.<Window>find("//Window[@caption='My Window']",
New FindOptions(False));
if (mainWindow){
    mainWindow.closeSynchron();
}
```

Identifying Multiple Objects with One Locator

You can use the `FindAll` method to identify all objects that match a locator rather than only identifying the first object that matches the locator.

Example

The following code example uses the `FindAll` method to retrieve all hyperlinks of a Web page:

```
List<DomLink> links = browserWindow.<DomLink>findAll("//a");
```

Locator Customization

This section describes how you can create stable locators that enable Silk4J to reliably recognize the controls in your application under test (AUT).

Silk4J relies on the identifiers that the AUT exposes for its UI controls and is very flexible and powerful in regards to identifying UI controls. Silk4J can use any declared properties for any UI control class and can also create locators by using the hierarchy of UI controls. From the hierarchy, Silk4J chooses the most appropriate items and properties to identify each UI control.

Silk4J can exclude dynamic numbers of controls along the UI control hierarchy, which makes the object recognition in Silk4J very robust against changes in the AUT. Intermediate grouping controls that change the hierarchy of the UI control tree, like formatting elements in Web pages, can be excluded from the object recognition.

Some UI controls do not expose meaningful properties, based on which they can be identified uniquely. Applications which include such controls are described as applications with *bad testability*. Hierarchies, and especially dynamic hierarchies, provide a good means to create unique locators for such applications. Applications with *good testability* should always provide a simple mechanism to identify UI controls uniquely.

One of the simplest and most effective practices to make your AUT easier to test is to introduce stable identifiers for controls and to expose these stable identifiers through the existing interfaces of the application.

Stable Identifiers

A *stable identifier* for a UI control is an identifier that does not change between invocations of the control and between different versions of the application, in which the UI control exists. A stable identifier needs to be unique in the context of its usage, meaning that no other control with the same identifier is accessible at the same time. This does not necessarily mean that you need to use GUID-style identifiers that are unique in a global context. Identifiers for controls should be readable and provide meaningful names. Naming conventions for these identifiers will make it much easier to associate the identifier to the actual control.

Example: Is the caption a good identifier for a control?

Very often test tools are using the *caption* as the default identifier for UI controls. The caption is the text in the UI that is associated with the control. However, using the caption to identify a UI control has the following drawbacks:

- The caption is not stable. Captions can change frequently during the development process. For example, the UI of the AUT might be reviewed at the end of the development process. This prevents introducing UI testing early in the development process because the UI is not stable.
- The caption is not unique. For example, an application might include multiple buttons with the caption **OK**.
- Many controls are not exposing a caption, so you need to use another property for identification.
- Using captions for testing localized applications is cumbersome, as you need to maintain a caption for a control in each language and you also have to maintain a complex script logic where you dynamically can assign the appropriate caption for each language.

Creating Stable Locators

One of the main advantages of Silk4J is the flexible and powerful object-recognition mechanism. By using XPath notation to locate UI controls, Silk4J can reliably identify UI controls that do not have any suitable

attributes, as long as there are UI elements near the element of interest that have suitable attributes. The XPath locators in Silk4J can use the entire UI control hierarchy or parts of it for identifying UI controls. Especially modern AJAX toolkits, which dynamically generate very complex Document Object Models (DOMs), do not provide suitable control attributes that can be used for locating UI controls.

In such a case, test tools that do not provide intelligent object-recognition mechanisms often need to use index-based recognition techniques to identify UI controls. For example, identify the *n*-th control with icon *Expand*. This often results in test scripts that are hard to maintain, as even minor changes in the application can break the test script.

A good strategy to create stable locators for UI controls that do not provide useful attributes is to look for an anchor element with a stable locator somewhere in the hierarchy. From that anchor element you can then work your way to the element for which you want to create the locator.

Silk4J uses this strategy when creating locators, however there might be situations in which you have to manually create a stable locator for a control.

Example: Locating Siblings of a Control

This topic describes how you can locate a control, which does not provide any meaningful attributes that can be used in locators, when a stable locator for a sibling of the control is available.

Assume that you have already identified the control **Item 0.0**, which has the following stable locator:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']
```

If you know that **Item 0.0** has a following-sibling of the type *a*, you can use the following code to build a stable locator for the sibling:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']/following-sibling::a
```

You can also use the sibling approach to identify text fields. Text fields often do not provide any meaningful attributes that can be used in locators. By using the label of a text field, you could create a meaningful locator for the text field, because the label is the best identifier for the text field from the perspective of a tester. You can easily use the label as a part of the locator for a test field by using the sibling approach. For example, if the text field is a preceding-sibling of a label with the text **User Name**, you can use the following locator:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='User Name']/preceding-sibling::input[@type='text']
```

Example: Locating the Expand Icon in a Dynamic GWT Tree

The Google Widget Toolkit (GWT) is a very popular and powerful toolkit, which is hard to test. The dynamic tree control is a very commonly used UI control in GWT. To expand the tree, we need to identify the **Expand** icon element.

You can find a sample dynamic GWT tree at <http://samples.gwtproject.org/samples/Showcase/Showcase.html#!CwTree>.

The default locator generated by Silk4J is the following:

```
/BrowserApplication//BrowserWindow//DIV[@id='gwt-debug-cwTree-dynamicTree-root-child0']/DIV/DIV[1]//IMG[@border='0']
```

For the following reasons, this default locator is no reliable locator for identifying the **Expand** icon for the control **Item 0.0**:

- The locator is complex and built on multiple hierarchies. A small change in the DOM structure, which is dynamic with AJAX, can break the locator.
- The locator contains an index for some of the controls along the hierarchy. Index based locators are generally weak as they find controls by their occurrence, for example finding the sixth expand icon in a tree does not define the control well. An exception to that rule would be if the index is used to express different data sets that you want to identify, for example the sixth data row in a grid.

Often a good strategy for finding better locators is to search for siblings of elements that you need to locate. If you find siblings with better locators, XPath allows you to construct the locator by identifying those siblings. In this case, the tree item **Item 0.0** provides a better locator than the **Expand** icon. The locator of the tree item **Item 0.0** is a stable and simple locator as it uses the `@textContent` property of the control.

By default, Silk4J uses the property `@id`, but in GWT the `@id` is often not a stable property, because it contains a value like `'gwt-uid- \langle nnn \rangle '`, where `\langle nnn \rangle` changes frequently, even for the same element between different calls.

You can manually change the locator to use the `@textContent` property instead of the `@id`.

Original Locator:

```
/BrowserApplication//BrowserWindow//DIV[@id='gwt-uid-109']
```

Alternate Locator:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']
```

Or you can instruct Silk4J to avoid using `@id='gwt-uid- \langle nnn \rangle '`. In this case Silk4J will automatically record the stable locator. You can do this by adding the text pattern that is used in `@id` properties to the locator attribute value blacklist. In this case, add `gwt-uid*` to the blacklist.

When inspecting the hierarchy of elements, you can see that the control **Item 0.0** and the **Expand** icon control have a joint root node, which is a `DomTableRow` control.

To build a stable locator for the **Expand** icon, you first need to locate **Item 0.0** with the following locator:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']
```

Then you need to go up two levels in the element hierarchy to the `DomTableRow` element. You express this with XPath by adding `../../` to the locator. Finally you need to search from `DomTableRow` for the **Expand** icon. This is easy as the **Expand** icon is the only `IMG` control in the sub-tree. You express this with XPath by adding `//IMG` to the locator. The final stable locator for the **Expand** icon looks like the following:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']/../..///IMG
```

Or even better, use the XPath ancestor axis to locate the **Expand** icon:

```
/BrowserApplication//BrowserWindow//DIV[@textContent='Item 0.0']/  
ancestor::tr//IMG
```

Custom Attributes

Many UI technologies provide a mechanism that allows them to extend the set of predefined attributes of UI controls with custom attributes. These custom attributes can be used by the application developer to introduce stable identifiers that uniquely identify the control. Silk4J can access custom attributes of UI controls and can also use these custom attributes to identify UI controls.

Using special automation for the identification of UI controls has several advantages compared to using the defined attributes like `caption`. Being able to establish stable identifiers in the application code and to expose these identifiers through either custom attributes or defined automation properties leads to understandable and maintainable test-automation scripts, allowing you to start with your test automation early in the development process.

You can configure the attributes used for identification by using the flexible locator strategy of Silk4J.

Custom Attributes for Apache Flex Applications

Apache Flex applications use the predefined property `automationName` to specify a stable identifier for the Apache Flex control as follows:

```
<?xml version="1.0" encoding="utf-8"?>  
  <s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"  
    xmlns:s="library://ns.adobe.com/flex/spark"
```

```

xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300">
<fx:Script>
...
</fx:Script>
<s:Button x="247" y="81" label="Button" id="button1" enabled="true"
click="button1_clickHandler(event)"
automationName="AID_buttonRepeat"/>
<s:Label x="128" y="123" width="315" height="18" id="label1"
verticalAlign="middle"
text="awaiting your click" textAlign="center"/>
</s:Group>

```

Apache Flex application locators look like the following:

```
...//SparkApplication//SparkButton[@caption='AID_buttonRepeat']
```



Attention: For Apache Flex applications, the *automationName* is always mapped to the locator attribute *caption* in Silk4J. If the *automationName* attribute is not specified, Silk4J maps the property *ID* to the locator attribute *caption*.

Java SWT Custom Attributes

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

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

Java SWT Example

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

```

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

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

```

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

```

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

```

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

Custom Attributes for Web Applications

HTML defines a common attribute *ID* that can represent a stable identifier. By definition, the *ID* uniquely identifies an element within a document. Only one element with a specific *ID* can exist in a document.

However, in many cases, and especially with AJAX applications, the *ID* is used to dynamically identify the associated server handler for the HTML element, meaning that the *ID* changes with each creation of the Web document. In such a case the *ID* is not a stable identifier and is not suitable to identify UI controls in a Web application.

A better alternative for Web applications is to introduce a new custom HTML attribute that is exclusively used to expose UI control information to Silk4J.

Custom HTML attributes are ignored by browsers and by that do not change the behavior of the AUT. They are accessible through the DOM of the browser. Silk4J allows you to configure the attribute that you want to use as the default attribute for identification, even if the attribute is a custom attribute of the control class. To set the custom attribute as the default identification attribute for a specific technology domain, click **Silk4J > Edit Options > Custom Attributes** and select the technology domain.

The application developer just needs to add the additional HTML attribute to the Web element.

Original HTML code:

```
<A HREF="http://abc.com/control=4543772788784322..." <IMG  
src="http://abc.com/xxx.gif" width=16 height=16> </A>
```

HTML code with the new custom HTML attribute *AUTOMATION_ID*:

```
<A HREF="http://abc.com/control=4543772788784322..."  
AUTOMATION_ID = "AID_Login" <IMG src="http://abc.com/xxx.gif"  
width=16 height=16> </A>
```

When configuring the custom attributes, Silk4J uses the custom attribute to construct a unique locator whenever possible. Web locators look like the following:

```
...//DomLink[@AUTOMATION_ID='AID_Login']
```

Example: Changing ID

One example of a changing ID is the Google Widget Toolkit (GWT), where the ID often holds a dynamic value which changes with every creation of the Web document:

```
ID = 'gwt-uid-<nnn>'
```

In this case `<nnn>` changes frequently.

Custom Attributes for Windows Forms Applications

Windows Forms applications use the predefined automation property `automationId` to specify a stable identifier for the Windows forms control.

Silk4J automatically will use this property for identification in the locator. Windows Forms application locators look like the following:

```
/FormsWindow//PushButton[@automationId='btnBasicControls']
```

Custom Attributes for WPF Applications

WPF applications use the predefined automation property `AutomationProperties.AutomationId` to specify a stable identifier for the WPF control as follows:

```
<Window x:Class="Test.MainWindow"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
Title="MainWindow" Height="350" Width="525">  
  <Grid>  
    <Button AutomationProperties.AutomationId="AID_buttonA">The  
Button</Button>  
  </Grid>  
</Window>
```

Silk4J automatically uses this property for identification in the locator. WPF application locators look like the following:

```
/WPFWindow[@caption='MainWindow']//WPFButton[@automationId='AID_buttonA']
```

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 7 in the sample tree, if you have a query like `Root.Find("//Node[@id='7']")`, replace it with a query like `Root.Find("/Node[@id='6']/Node[@id='7']")`. The first query searches the element tree for the elements with the identifiers 1 to 7. The second query searches only for the element first level nodes, which are the node with the identifier 1 and the node with the identifier 6, for the node with the identifier 6, and then searches in the subtree of the node with the identifier 6 for the first leaf with the identifier 7.
- When you want to locate multiple items in the same hierarchy, first locate the hierarchy, and then locate the items based on their common root node. If you have a query like `Root.FindAll("/Node[@id='1']/Leaf")`, replace it with a query like the following:

```
public void test() {
    TestObject commonRootNode = desktop.find("//Node[@id='1']");
    commonRootNode.find("/Leaf[@id='2']");
    commonRootNode.find("/Leaf[@id='3']");
    commonRootNode.find("/Leaf[@id='4']");
    commonRootNode.find("/Leaf[@id='5']");
}
```

Locator Spy

You can use the **Locator Spy** to record unique Silk Test locators or WebDriver locators for any control in your application under test (AUT). From the **Locator Spy**, you can copy the locator for a control or any attributes of the control into methods in your scripts. You can also use the **Locator Spy** to edit the attributes of the locator for a control and you to validate these changes. Using the **Locator Spy** ensures that the locator for a control is valid.

The object tree in the **Locator Spy** lists all the controls that are available in the AUT. You can use the object tree to inspect the available controls and the control hierarchy of the AUT. When recording WebDriver locators, a ">" in the object tree denotes switching from one IFrame to another.



Note: The locator attributes table of the **Locator Spy** displays all attributes that you can use in the locator. For web applications, the table also includes any attributes that you have defined to be ignored during recording.

Object Maps

An object map is a test asset that contains items that associate a logical name (an alias) with a control or a window, rather than the control or window's locator. Once a control is registered in an object map asset, all references to it in scripts are made by its alias, rather than by its actual locator name.

You can use object maps to store objects that you are using often in multiple scripts. Multiple tests can reference a single object map item definition, which enables you to update that object map definition once and have Silk4J update it in all tests that reference the object map definition.

In your scripts, you can mix object map identifiers and locators. This feature enables you to keep your object maps relatively small and easier to manage. You can simply store the commonly used objects in your object maps, and use locators to reference objects that are rarely used.



Tip: To optimally use the functionality that object maps provide, create an individual project in Silk4J for each application that you want to test.

Example for object maps

The following construct shows a definition for a `BrowserWindow` where the locator is used:

```
_desktop.BrowserApplication("cnn_com").BrowserWindow("//  
BrowserWindow[1]")
```

The name of the object map asset is `cnn_com`. The locator that can be substituted by an alias in the object map is the following:

```
"//BrowserWindow[1]"
```

The object map entry for the `BrowserWindow` is `BrowserWindow`.

The resulting definition of the `BrowserWindow` in the script is the following:

```
_desktop.BrowserApplication("cnn_com").BrowserWindow("BrowserWin  
dow")
```

If the index in the locator changes, you can just change the alias in the object map, instead of having to change every appearance of the locator in your test script. Silk4J will update all tests that reference the object map definition.

Example for mixing object map identifiers and locators

The following sample code shows how you can mix object map identifiers and locators to specify a rarely used child object of an object stored in an object map:

```
Window window = _desktop.find("MyApplication"); // object map  
id - the application window is used often  
MenuItem aboutMenuItem = _desktop.find("//  
MenuItem[@caption='About']"); // locator - the About dialog is  
only used once  
aboutMenuItem.select();
```

Advantages of Using Object Maps

Object maps have the following advantages:

- They simplify test maintenance by applying changes made to a locator for an object map item to all tests that include the corresponding object map item.
- They ease the handling of locators in a large scale functional testing environment.
- They can be managed independent of individual scripts.
- They substitute complex locator names with descriptive names, which can make scripts easier to read.
- They eliminate dependence on locators, which may change if the test application is modified.

Turning Object Maps Off and On

You can configure Silk4J to use the locator name or the alias from the object map during recording.

To use the alias from the object map during recording:

1. Click **Silk4J > Edit Options**.
2. Click **Recording**.
3. To define whether you want to record object map entries or XPath locators, select the appropriate recording mode from the **OPT_RECORD_OBJECTMAPS_MODE** list:
 - **Object map entries for new and existing objects**. This is the default mode.
 - **XPath locators for new and existing objects**.
 - **XPath locators for new objects only**. For objects that already exist in an object map, the object map entry is reused. Choosing this setting enables you to create object maps for the main controls of an AUT, and to persist these object maps while creating additional tests against the AUT.



Note: In addition to the XPath attributes, Silk4J uses additional attributes of the element when merging object maps during locator recording. However, attributes that might lead to ambiguous usage of object map IDs in a recorded script are not used to map locators to existing object map entries.



Note: When you enable the **Record object maps** setting, object map item names display in place of locators throughout Silk4J. For instance, if you view the **Application Configurations** category in the **Properties** pane, you will notice that the **Locator** box shows the object map item name rather than the locator name.

Using Assets in Multiple Projects

In Silk4J, image assets, image verifications, and object maps are referred to as *assets*. If you want to use assets outside of the scope of the project in which they are located, you need to add a direct project dependency from the project in which you want to use the assets to the project in which the assets are located. When you are playing back tests from Eclipse, all dependent projects are added to the classpath for the test execution, and therefore Silk4J can find the assets in the dependent projects.

During replay, when an asset is used, Silk4J firstly searches in the *current project* for the asset. The current project is the JAR file which contains the test code that is currently executed. If Silk4J does not find the asset in the current project, Silk4J additionally searches all other projects in the classpath.. If the asset is still not found, Silk4J throws an error.

If assets with the same name exist in more than one project, and you do not want to use the asset that is included in the current project, you can define which specific asset you want to use in any method that uses the asset. To define which asset you want to use, add the asset namespace as a prefix to the asset name when calling the method. The asset namespace defaults to the project name.



Note: When you start working with Silk4J, the asset namespace option is added to the `silk4j.settings` file of every Silk4J project in your workspace that has been created with a previous version of Silk4J.

Example: Adding a project dependency

If the project *ProjectA* contains a test that calls the following code:

```
window.imageClick("imageAsset");
```

and the image asset *imageAsset* is located in project *ProjectB*, you need to add a direct project dependency from *ProjectA* to *ProjectB*.

To add a project dependency in Eclipse, right-click the project and select **Properties**. Select **Java Build Path**, click on the **Projects** tab, and add your project here.



Note: Using **Project References** instead of **Java Build Path** does not work.

Example: Calling a specific asset

If *ProjectA* and *ProjectB* both contain an image asset with the name *anotherImageAsset*, and you explicitly want to click the image asset from *ProjectB*, use the following code:

```
window.imageClick("ProjectB:anotherImageAsset")
```

Merging Object Maps During Action Recording

When you record actions with Silk4J, Silk4J checks if existing object map entries can be reused. Silk4J checks this directly during recording, when a new locator is generated. Silk4J checks if the object that is currently recorded in the application under test exactly matches an existing object map entry, and if yes, Silk4J reuses the object map identifier from the object map.

This behavior has the following benefits:

- Silk4J correctly reuses an object map identifier during recording, even if the locator in the object map has changed.
- A recorded script cannot contain wrong object map identifiers, and therefore will never fail to play back because of a wrong object map identifier.
- If you restructure your object map, for example by adding an additional level of hierarchy, the object map identifiers are still reused.

Example

Silk4J records the following script when you click on the **Products** link in the Micro Focus website, <http://www.borland.com>.

```
With _desktop.BrowserApplication( "borland_com" )  
  With .BrowserWindow( "BrowserWindow" )  
    .DomLink( "Products" ).Click( MouseButton .Left, New Point  
(47, 18))  
  End With  
End With
```

The recorded object map looks like this:

```
borland_com //BrowserApplication  
  BrowserWindow //BrowserWindow  
    Products //  
A[@textContents='Products']
```

You could now manually restructure the object map to include the header section of the Micro Focus website:

```
borland_com //BrowserApplication  
  BrowserWindow //BrowserWindow
```

```
header //
HEADER[@role='banner']
  Products //
A[@textContent='Products']
```

When you now record a click on the **Products** link the object map is reused correctly, and the following script is recorded:

```
With _desktop.BrowserApplication( "borland_com" )
  With .BrowserWindow( "BrowserWindow" )
    .DomElement("header").DomLink( "Products" ).Click( MouseButton.Left, New Point (47, 18))
  End With
End With
```



Note: When you record another object in the header section of the Micro Focus website, for example the **About** link, Silk4J adds the **About** object map entry as a child of **BrowserWindow**, and not of **header**.

Using Object Maps with Web Applications

By default, when you record actions against a Web application, Silk4J creates an object map with the name *WebBrowser* for native browser controls and an object map asset for every Web domain.

For common browser controls which are not specific for a Web domain, like the main window or the dialog boxes for printing or settings, an additional object map is generated in the current project with the name *WebBrowser*.

In the object map, you can edit the URL pattern by which the object map entries are grouped. When you edit the pattern, Silk4J performs a syntactical validation of the pattern. You can use the wildcards * and ? in the pattern.

Example

When you record some actions on <http://www.borland.com> and <http://www.microfocus.com> and then open the printer dialog, the following three new object map assets are added to the **Asset Browser**:

- WebBrowser
- borland_com
- microfocus_com



Note: Silk4J generates the new object map assets only for projects without an object map. If you record actions against a Web application for which Silk4J already includes an object map that was generated with a version of Silk4J prior to version 14.0, the additionally recorded entries are stored into the existing object map, and there are no additional object map assets generated for the Web domains.

Renaming an Object Map Item

You can manually rename items and locators in an object map.



Warning: Renaming an object map item affects every script that uses that item. For example, if you rename the **Cancel** button object map item from **CancelMe** to **Cancel**, every script that uses **CancelMe** must be changed manually to use **Cancel**.

Object map items must be unique. If you try to add a duplicate object map item, Silk4J notifies you that the object must be unique.

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error. Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,]. Invalid locator paths include: empty or incomplete locator paths.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:
 - Double-click the object map that includes the object map item that you want to rename.
 - Right-click the object map that includes the object map item that you want to rename and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Navigate to the object map item that you want to rename.
For example, you might need to expand a node to locate the item that you want to rename.
4. Click the object that you want to rename and then click the object again.
5. Type the item name that you want to use and then press **Enter**.
If you use an invalid character, the item name displays in red.
The new name displays in the **Item name** list.
6. Press **CTRL+S** to save your changes



Note: All child nodes of any node in the object map tree are sorted alphabetically when you save the object map.

If any existing scripts use the item name that you changed, you must manually change the scripts to use the new item name.



Note: While recording against a web application or a mobile web app, you can directly change the name of the object map entry in the **Choose Action** dialog. Right-click on the object and then expand the **Object identification** area of the **Choose Action** dialog. Then you can edit the object map entry in the **Object Map ID** field. This functionality is available if you are testing against one of the following browsers:

- Microsoft Edge.
- Apple Safari.
- Mozilla Firefox 41 or later.
- Google Chrome 50 or later.
- A mobile browser.

Modifying Object Maps

An existing object map is able to reuse existing object map identifiers during recording, even if you have added additional structural elements to the object map.

Example: Adding a DIV to an existing object map

Let us suppose you want to add a DIV element to bundle the email and login fields in the following simple object map:

```
demo_borland_com //
BrowserApplication
  BrowserWindow //
BrowserWindow
  login-form email //
INPUT[@id='login-form:email']
```

```

login-form login //
INPUT[@id='login-form:login']

You can change the structure of the object map by adding the new DIV loginArea and
the object map will still be able to correctly reuse the object map identifiers during
recording.

demo_borland_com //
BrowserApplication
  BrowserWindow //
BrowserWindow

loginArea // 'DIV[@id='
login']
  login-form email //
INPUT[@id='login-form:email']
  login-form login //
INPUT[@id='login-form:login']

```

Modifying a Locator in an Object Map

Locators are automatically associated with an object map item when you record a script. However, you might want to modify a locator path to make it more generic. For example, if your test application automatically assigns the date or time to a specific control, you might want to modify the locator for that control to use a wildcard. Using a wildcard enables you to use the same locator for each test even though each test inserts a different date or time.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

- Double-click the object map that includes the locator that you want to modify.
- Right-click the object map that includes the locator that you want to modify and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Navigate to the locator that you want to modify.
For example, you might need to expand a node to locate the locator that you want to modify.
4. Click the locator path that you want to modify and then click the locator path again.
5. If you have a valid locator path, you can type the item name and locator path that you want to use and then press **Enter**. To determine a valid locator path, use the **Locator Spy** dialog box as described in the following steps:
 - a) In the Silk4J tool bar, click **Locator Spy**.
 - b) Position the mouse over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
 - c) Select the locator that you want to use in the **Locator Details** table.
 - d) Copy and paste the locator into the object map.
6. If necessary, modify the item name or locator text to meet your needs.
If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

Invalid locator paths include: empty or incomplete locator paths.

7. Press **CTRL+S** to save your changes

If any existing scripts use the locator path that you modified, you must manually change the visual tests or scripts to use the new locator path.

Updating Object Maps from the Test Application

If items in the test application change, you can use the **Object Map** UI to update the locators for these items.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:
 - Double-click the object map that you want to use.
 - Right-click the object map that you want to use and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Click **Update Locator**. The **Locator Spy** displays and Silk4J opens the test application.
4. Position the mouse cursor over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
5. Select the locator that you want to use in the **Locator Details** table.
6. Remove any attributes that you do not want to use from the locator that is displayed in the **Locator** text field.
7. Click **Validate Locator** to validate that the locator works.
8. Click **Paste Locator to Editor** to update the locator in the object map.
9. Save the changed object map.

When you update an object map item from the AUT, you can change only the XPath representations of leaf nodes in the object map tree. You cannot change the XPath representations of any parent nodes. When the XPath representations of higher-level nodes in the object map tree are not consistent after the update, an error message displays.

Example

For example, suppose you have an object map item with an object map ID that has the following three hierarchy levels:

```
WebBrowser.Dialog.Cancel
```

The corresponding XPath representation of these hierarchy levels is the following:

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

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//Dialog`
- Third hierarchy level: `//PushButton[@caption='Cancel']`

You can use the following locator to update the object map item:

```
/BrowserApplication//Dialog//PushButton[@id='123']
```

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//Dialog`
- Third hierarchy level: `//PushButton[@id='123']`

You cannot use the following locator to update the object map item, because the second level hierarchy nodes do not match:

```
/BrowserApplication//BrowserWindow//PushButton[@id='9999999']
```

- First hierarchy level: `/BrowserApplication`
- Second hierarchy level: `//BrowserWindow`

- Third hierarchy level: `//PushButton[@id='9999999']`

Copying an Object Map Item

You can copy and paste object map entries within or between object maps. For example, if the same functionality exists in two separate test applications, you might copy a portion of one object map into another object map.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:
 - Double-click the object map that includes the object map item that you want to copy.
 - Right-click the object map that includes the object map item that you want to copy and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Navigate to the object map item that you want to copy.
For example, you might need to expand a node to locate the item that you want to copy.
4. Choose one of the following:
 - Right-click the object map item that you want to copy and choose **Copy tree**.
 - Click the object map item that you want to copy and then press `Ctrl+C`.
5. In the object map hierarchy, navigate to the position where you want to paste the item that you copied.
For instance, to include an item on the first level of the hierarchy, click the first item name in the item list. To position the copied item a level below a specific item, click the item that you want to position the copied item below.
To copy and paste between object maps, you must exit the map where you copied the object map item and open and edit the object map where you want to paste the object map item.
6. Choose one of the following:
 - Right-click the position in the object map where you want to paste the copied object map item and choose **Paste**.
 - Click the position in the object map where you want to paste the copied object map item and then press `Ctrl+V`.

The object map item displays in its new position in the hierarchy.

7. Press **CTRL+S** to save your changes

If any existing scripts use the object map item name that you moved, you must manually change the scripts to use the new position in the hierarchy.

Adding an Object Map Item

Object map items are automatically created when you record a script. Occasionally, you might want to manually add an object map item.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Double-click the object map to which you want to add the new item. The object map displays a hierarchy of the object map items and the locator associated with each item.
3. In the object map hierarchy, right-click on the item below which you want to add the new object map item.

For instance, to include an item on the first level of the hierarchy, right-click on the first item name in the item list. To position the new item a level below a specific item, right-click on the item below which you want to position the new item.

4. Click **Insert new**. A new item is added to the hierarchy, as the first child of the current node.
5. If you have a valid locator path, you can type the item name and locator path that you want to use and then press **Enter**. To determine a valid locator path, use the **Locator Spy** dialog box as described in the following steps:
 - a) In the Silk4J tool bar, click **Locator Spy**.
 - b) Position the mouse over the object that you want to record and press **CTRL+ALT**. Silk4J displays the locator string in the **Locator** text field.
 - c) Select the locator that you want to use in the **Locator Details** table.
 - d) Copy and paste the locator into the object map.
6. If necessary, modify the item name or locator text to meet your needs.

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

Invalid locator paths include: empty or incomplete locator paths.
7. Press **CTRL+S** to save your changes



Note: All child nodes of any node in the object map tree are sorted alphabetically when you save the object map.

Opening an Object Map from a Script

When you are editing a script, you can open an object map by right clicking on an object map entry in the script and selecting **Open Silk4JAsset**. This will open the object map in the GUI.

Use **Ctrl+Click** and click on an object map entry and the object map entry will turn into a hyperlink. Click it to open it.

Example

```
@Test
public void test() {
    Window mainWindow = desktop.<Window>find("Untitled -
Notepad");
    mainWindow.<TextField>find("TextField").typeKeys("hello");
}
```

In the previous code sample, right-click `Untitled - Notepad` to open the entry `Untitled - Notepad` in the object map, or right-click `TextField` to open the entry `Untitled - Notepad.TextField` in the object map.

Highlighting an Object Map Item in the Test Application

After you add or record an object map item, you can click **Highlight** to highlight the item in the test application. You might want to highlight an item to confirm that it's the item that you want to modify in the object map.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

- Double-click the object map that you want to use.
- Right-click the object map that you want to use and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. In the object map hierarchy, select the object map item that you want to highlight in the test application.



Note: Ensure that only one instance of the test application is running. Running multiple instances of the test application will cause an error because multiple objects will match the locator.

4. Click **Highlight**.

The **Select Application** dialog box might open if the test application has not been associated with the object map. If this happens, select the application that you want to test and then click **OK**.

Silk4J opens the test application and displays a green box around the control that the object map item represents.

Finding Errors in an Object Map

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error. Use the toolbar in the **Object Map** window to navigate to any errors.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:

- Double-click the object map that you want to troubleshoot.
- Right-click the object map that you want to troubleshoot and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Look for any item name or locator text displayed in red.
4. If necessary, modify the item name or locator text to meet your needs.

If you use an invalid character or locator, the item name or locator text displays in red and a tooltip explains the error.

Invalid characters for object map items include: \, /, <, >, ", :, *, ?, |, =, ., @, [,].

Invalid locator paths include: empty or incomplete locator paths.

5. Press **CTRL+S** to save your changes

Deleting an Object Map Item

You might want to delete an item from an object map if it no longer exists in the test application or for some other reason.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Double-click the object map that includes the object map item that you want to delete. The object map displays a hierarchy of the object map items and the locator associated with each item.
3. Navigate to the object map item that you want to delete.
For example, you might need to expand a node to locate the object map item that you want to delete.
4. Choose one of the following:
 - Right-click the object map item that you want to delete and choose **Delete**, or choose **Delete tree** to additionally delete all child items of the object map item.
 - Click the object map item that you want to delete and then press **DEL**, or press **CTRL+DEL** to additionally delete all child items of the object map item.

After deleting an object map item, the focus moves to the next item in the object map.

5. Press **CTRL+S** to save your changes

If any existing scripts use the object map item or its children that you deleted, you must manually change any references to that object map item in the scripts.

Initially Filling Object Maps

As a best practice, we recommend that you fill your object map and then review all object map items before you record your tests.

To initially fill your object map with all available items in the AUT, you might create a test that clicks every object and opens every window and dialog box in your test application. Then, you can review the object map item for each object and make any necessary modifications before you record your functional tests. After you have reviewed and modified the object map items you can delete the test that you have created to fill the object map.



Tip: You can use the arrow keys to navigate between items in an object map.

Grouping Elements in Object Maps

When items in an object map have no consistent parent object, you can group these elements by adding a new tree item with the locator ".", which is the locator for the current element in Xpath.



Warning: Grouping object map items affects every script that uses these items. Every script that uses these items must be changed manually to use the new locators.

1. In the **Package Explorer**, click on the **Object Maps** folder of the project in which the object map that you want to change is located.
2. Choose one of the following:
 - Double-click the object map that you want to edit.
 - Right-click the object map that you want to edit and choose **Open**.

The object map displays a hierarchy of the object map items and the locator associated with each item.

3. Right click on the tree item below which you want to add the new structuring item and choose **Insert New**.
4. Double click the **Item name** field of the new object map item.
5. Type the item name that you want to use and then press **Enter**.
If you use an invalid character, the item name displays in red.
The new name displays in the **Item name** list.
6. Click the **Locator path** field of the new object map item and type . into the field.
7. Press **Enter**.
8. For every object map item that you want to relocate to a new location under the new item:
 - a) Right click on the item that you want to relocate and choose **Cut tree**.
 - b) Right click on the new structuring item and choose **Paste**.
9. Press **CTRL+S** to save your changes

Object Maps: Frequently Asked Questions

This section lists questions that you might encounter when using object maps with Silk4J.

Can I Merge Multiple Object Maps Into a Single Map?

Although Micro Focus recommends recording into the same object map instead of merging existing object maps, you can use a text editor to merge multiple object maps into a single map.

What Happens to an Object Map when I Delete a Test Script?

When you delete a test script that includes object map entries, the associated object maps are not changed. All object map entries are persisted.

Can I Manually Create an Object Map for My Application Under Test?

Micro Focus recommends creating object maps during the recording of a test. However, you can also create an empty object map and manually add object map entries to this map.

To create a new object map, select **Silk4J > New Object Map** from the menu.

Image Recognition Support

You can use image recognition in the following situations:

- To conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *image clicks* instead of coordinate-based clicks to click on a specified image.
- To test graphical objects in the application under test, for example charts.
- To perform a check of the visible UI of the application under test.

If you want to click on a control that is otherwise not recognizable, you can use the `imageClick` method with an image asset. If you want to verify that an otherwise not recognizable control exists in your application under test, you can use the `verifyAsset` method with an image verification.

Image recognition methods are supported for all technology domains that are supported by Silk4J.

 **Note:** Image recognition methods do not work with controls that are not visible on the screen. For example, you cannot use image recognition for an image that is scrolled out of view.

Image Click Recording

Image click recording is disabled by default in favor of coordinate-based click recording, because image click recording might generate a confusingly large number of images.

To enable image click recording, click **Silk4J > Edit Options**, select the **Recording** tab, and check the check box in the **Record image clicks** section.

 **Note:** When recording on a mobile browser, you do not have to enable image click recording.

When image click recording is enabled, Silk4J records `ImageClick` methods when object recognition or text recognition is not possible. You can insert image clicks in your script for any control, even if the image clicks are not recorded.

If you do not wish to record an `ImageClick` action, you can turn off image click recording and record normal clicks or text clicks.

 **Note:** The recorded images are not reused. Silk4J creates a new image asset for each image click that you record.

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

Image Recognition Methods

Silk4J provides the following methods for image recognition:

Method	Description
<code>imageClick</code>	Clicks in the middle of the image that is specified in an asset. Waits until the image is found or the <i>Object resolve timeout</i> , which you can define in the synchronization options, is over.
<code>imageExists</code>	Returns whether the image that is specified in an asset exists.

Method	Description
<code>imageRectangle</code>	Returns the object-relative rectangle of the image that is specified in an asset.
<code>imageClickFile</code>	Clicks on the image that is specified in a file.
<code>imageExistsFile</code>	Returns whether the image that is specified in a file exists.
<code>imageRectangleFile</code>	Returns the object-relative rectangle of the image that is specified in a file.
<code>verifyAsset</code>	Executes a verification asset. Throws a <code>VerificationFailedException</code> if the verification does not pass.
<code>tryVerifyAsset</code>	Executes a verification asset and returns whether the verification passed.

 **Note:** Image recognition methods do not work with controls that are not visible on the screen. For example, you cannot use image recognition for an image that is scrolled out of view.

Image Assets

You can use image assets in the following situations:

- To conveniently interact with test applications that contain highly customized controls, which cannot be identified using object recognition. You can use *image clicks* instead of coordinate-based clicks to click on a specified image.
- To test graphical objects in the application under test, for example charts.

Image assets consist of an image with some additional information that is required by Silk4J to work with the asset.

Silk4J provides the following methods for image assets:

Method	Description
<code>imageClick</code>	Clicks in the middle of the specified image asset. Waits until the image is found or the <i>Object resolve timeout</i> , which you can define in the synchronization options, is over.
<code>imageExists</code>	Returns whether the specified image asset exists.
<code>imageRectangle</code>	Returns the object-relative rectangle of the specified image asset.

Image assets must be located in the `Image Assets` folder of the project. The `.imageasset` files must be embedded resources.

Creating an Image Asset

You can create image assets in one of the following ways:

- By inserting a new image asset into an existing script.
- During recording.
- From the menu.

To create a new image asset from the menu, perform the following steps:

1. In the menu, click **Silk4J > New Image Asset**.
2. Select the project, to which you want to add the new image asset, and type a meaningful name for the asset into the **Name** field.

3. Click **Finish**. The image asset UI opens.
4. Select how you want to add an image to the asset.
 - If you want to use an existing image, click **Browse** and select the image file.
 - If you want to capture a new image from the UI of the application under test, select **Capture**. If you are testing a Web application, you can select the browser on which you want to capture the image from the **Select Browser** window.
 - If you want to capture a new image after a delay of three seconds, for example to expand a menu in the application under test before the image is captured, select **Capture with Pause**.
5. If you have selected to capture a new image, select the area of the screen that you want to capture and click **Capture Selection**.
6. *Optional*: Click **Verify** to check if Silk4J can find the image asset in the UI of the AUT.
If you are testing a Web application, you can select the browser on which you want to capture the image from the **Select Browser** window.
7. *Optional*: Check the **Click position** check box to select the location on which any clicks on the image asset are performed.

The default location is the center of the image. Type the location into the **x** and **y** fields or select the location on the image.

8. Specify the **Accuracy Level**.

The accuracy level defines how much the image to be verified is allowed to be different to the image in the application under test, before Silk4J declares the images as different. This is helpful if you are testing multiple systems or browsers with different screen resolutions. We recommend to choose a high level of accuracy in order to prevent false positives. You can change the default accuracy level in the options.



Note: When you set the **Accuracy Level** to less than five, the actual colors of the images are no longer considered for the comparison. Only the grayscale representations of the images are compared.

9. Save the image asset.

The new image asset is listed under the current project in the **Package Explorer**, and you can use it to perform image clicks.

You can add multiple images to the same image asset.



Note: To add an image click while recording against a mobile browser, you can right-click in the **Recording** window and select **ImageClick** from the action list.

Adding Multiple Images to the Same Image Asset

During testing, you will often need to test functionality on multiple environments and with different testing configurations. In a different environment, the actual image might differ in such a degree from the image that you have captured in the image asset, that image clicks might fail, although the image is existing. In such a case, you can add multiple images to the same image asset.

To add an additional image to an image asset:

1. Double-click on the image asset to which you want to add an additional image. The image asset UI opens.
2. Click on the plus sign in the lower part of the UI to add a new image to the image asset.
3. Save the image asset.

The new image is added to the asset. Each time an image click is called, and until a match is achieved, Silk4J will compare the images in the asset with the images in the UI of the application under test. By default, Silk4J compares the images in the order in which they have been added to the asset.



Note: To change the order in which Silk4J compares the images, click on an image in the lower part of the image asset UI and drag the image to the position that you want. The order lowers from left to right. The image that is compared first is the image in the left-most position.

Opening an Asset from a Script

When you are editing a script, you can open an asset by right clicking it and selecting **Open Silk4JAsset**. This will open the asset in the GUI.

If the asset is a reference to a file on the system, for example, referenced by `ImageClickFile`, the file will be opened by your system's default editor.

Use `Ctrl+Click` and click on an asset and the asset will turn into a hyperlink. Click it to open it.

Image Verifications

You can use an *Image Verification* to check if an image exists in the UI of the application under test (AUT) or not.

Image verifications consist of an image with some additional information that is required by Silk4J to work with the asset.

To execute an image verification, use the `verifyAsset` method.

Image verification assets must be located in the `Verifications` folder of the project. The `.verification` files must be embedded resources.

An image verification fails when Silk4J cannot find the image in the AUT. In this case the script breaks execution and throws a `VerificationFailedException`. To avoid this behavior, use the `tryVerifyAsset` method.

If the locator for the image verification is not found in the AUT, Silk4J throws an `ObjectNotFoundException`.

You can open a successful image verification in TrueLog Explorer by clicking **Open Verification** in the **Info** tab of the verification step. You can open a failed image verification in TrueLog Explorer by clicking **Show Differences** in the **Info** tab of the verification step. If a failed image verification would have been successful if a lower accuracy level had been used, the accuracy level that would have succeeded is suggested.

Creating an Image Verification

You can create image verifications in one of the following ways:

- By using the menu.
- During recording.

To create a new image verification in the menu, perform the following steps:

1. Click **Silk4J > New Image Verification**.
2. Select the project, to which you want to add the new image verification, and type a meaningful name for the verification into the **Name** field.
3. Click **Finish**. The image verification UI opens.
4. Click **Identify** to identify the image that you want to verify in the application under test.
5. *Optional:* If you want to recapture the same image from the application under test, because there is a change in comparison to the image that you had initially captured, click **Recapture**.

If you are testing a Web application, you can select the browser on which you want to capture the image from the **Select Browser** window.

6. *Optional:* You can click **Verify** to test if the image verification works. Silk4J searches for the image in the UI of the AUT, top-down and left to right, and highlights the first matching image.
7. *Optional:* You can add an exclusion area to the image verification, which will not be considered when Silk4J compares the image verification to the UI of the application under test (AUT).
8. *Optional:* You can set the option **Client Area Only** to define that only the part of the image that is actually part of the AUT is considered when Silk4J compares the image verification to the UI of the AUT.
9. Specify the **Accuracy Level**.

The accuracy level defines how much the image to be verified is allowed to be different to the image in the application under test, before Silk4J declares the images as different. This is helpful if you are testing multiple systems or browsers with different screen resolutions. We recommend to choose a high level of accuracy in order to prevent false positives. You can change the default accuracy level in the options.



Note: When you set the **Accuracy Level** to less than five, the actual colors of the images are no longer considered for the comparison. Only the grayscale representations of the images are compared.

10. Save the image verification.

The new image verification is listed in the **Package Explorer**, and you can use it to check if the image exists in the UI of your application under test.

Adding an Image Verification During Recording

You can add image verifications to your scripts to check if controls which are otherwise not recognizable exist in the UI of the application under test. To add an image verification during the recording of a script, perform the following steps:

1. Begin recording.
2. Move the mouse cursor over the image that you want to verify and click **Ctrl + Alt**. Silk4J asks you if you want to verify a property or an image.
3. Select **Create or Insert an Image Verification**.
4. Perform one of the following steps:
 - To create a new image verification in the image verification UI, select **New** from the list box.
 - To insert an existing image verification asset, select the image verification asset from the list box.
5. Click **OK**.
 - If you have chosen to create a new image verification, the image verification UI opens.
 - If you have chosen to use an existing image verification, the image verification is added to your script. You can skip the remaining steps in this topic.
6. To create a new image verification, click **Verify** in the image verification UI.
7. Move the mouse cursor over the image in the AUT and click **CTRL+ALT**. The image verification UI displays the new image verification.
8. Click **OK**. The new image verification is added to the current project.
9. Continue recording.

Using Assets in Multiple Projects

In Silk4J, image assets, image verifications, and object maps are referred to as *assets*. If you want to use assets outside of the scope of the project in which they are located, you need to add a direct project dependency from the project in which you want to use the assets to the project in which the assets are located. When you are playing back tests from Eclipse, all dependent projects are added to the classpath for the test execution, and therefore Silk4J can find the assets in the dependent projects.

During replay, when an asset is used, Silk4J firstly searches in the *current project* for the asset. The current project is the JAR file which contains the test code that is currently executed. If Silk4J does not find the asset in the current project, Silk4J additionally searches all other projects in the classpath.. If the asset is still not found, Silk4J throws an error.

If assets with the same name exist in more than one project, and you do not want to use the asset that is included in the current project, you can define which specific asset you want to use in any method that uses the asset. To define which asset you want to use, add the asset namespace as a prefix to the asset name when calling the method. The asset namespace defaults to the project name.



Note: When you start working with Silk4J, the asset namespace option is added to the `silk4j.settings` file of every Silk4J project in your workspace that has been created with a previous version of Silk4J.

Example: Adding a project dependency

If the project *ProjectA* contains a test that calls the following code:

```
window.imageClick("imageAsset");
```

and the image asset *imageAsset* is located in project *ProjectB*, you need to add a direct project dependency from *ProjectA* to *ProjectB*.

To add a project dependency in Eclipse, right-click the project and select **Properties**. Select **Java Build Path**, click on the **Projects** tab, and add your project here.



Note: Using **Project References** instead of **Java Build Path** does not work.

Example: Calling a specific asset

If *ProjectA* and *ProjectB* both contain an image asset with the name *anotherImageAsset*, and you explicitly want to click the image asset from *ProjectB*, use the following code:

```
window.imageClick("ProjectB:anotherImageAsset");
```

Enhancing Tests

This section describes how you can enhance a test.

Recording Additional Actions Into an Existing Test

Once a test is created, you can open the test and record additional actions to any point in the test. This allows you to update an existing test with additional actions.

1. Open an existing test script.
2. Select the location in the test script into which you want to record additional actions.



Note: Recorded actions are inserted after the selected location. The application under test (AUT) does not return to the base state. Instead, the AUT opens to the scope in which the preceding actions in the test script were recorded.

3. Click **Record Actions**.

Silk4J minimizes and the **Recording** window opens.

4. Record the additional actions that you want to perform against the AUT.

For information about the actions available during recording, see *Actions Available During Recording*.

5. To stop recording, click **Stop** in the **Recording** window.

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: <ul style="list-style-type: none">• int• INT• long• LONG• DWORD• BOOL• WPARAM• HWND The Java type <code>int</code> 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 <code>long</code> and <code>int64</code> . The Java type <code>long</code> 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 <code>short</code> and <code>WORD</code> . The Java type <code>short</code> 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 <code>bool</code> .
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 <code>double</code> .
com.borland.silktest.jtf.Rect	Use this for arguments with the C data type <code>RECT</code> . <code>Rect</code> cannot be used as a return value.
com.borland.silktest.jtf.Point	Use this for arguments with the C data type <code>POINT</code> . <code>Point</code> cannot be used as a return value.
com.borland.silktest.jtf.TestObject	Use this for arguments with the C data type <code>HWND</code> . <code>TestObject</code> cannot be used as a return value, however you can declare DLL functions that return a <code>HWND</code> with an <code>Integer</code> as the return type.  Note: The passed <code>TestObject</code> must implement the <code>com.borland.silktest.jtf.INativeWindow</code> 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 InOutArgument with a String in that is long enough to hold the resulting String.

Example

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

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

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

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

Aliasing a DLL Name

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

Example

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

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

Conventions for Calling DLL Functions

The following calling conventions are supported when calling DLL functions:

- `__stdcall`
- `__cdecl`

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

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

Example

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

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

Custom Controls

Silk4J provides the following features to support you when you are working with custom controls:

- The *dynamic invoke* functionality of Silk4J enables you to directly call methods, retrieve properties, or set properties on an actual instance of a control in the application under test (AUT).
- The *class mapping* functionality enables you to map the name of a custom control class to the name of a standard Silk Test class. You can then use the functionality that is supported for the standard Silk Test class in your test.

Silk4J supports managing custom controls over the UI for the following technology domains:

- Win32
- Windows Presentation Foundation (WPF)
- Windows Forms
- Java AWT/Swing
- Java SWT
- You can add code to the AUT to test custom controls.
- The **Manage Custom Controls** dialog box enables you to specify a name for a custom control that can be used in a locator and also enables you to write reusable code for the interaction with the custom control.



Note: For custom controls, you can only record methods like `click`, `textClick`, and `typeKeys` with Silk4J. You cannot record custom methods for custom controls except when you are testing Apache Flex applications.

Dynamic Invoke

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

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

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

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

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

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

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

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

 **Note:** You cannot dynamically invoke methods for DOM elements.

Frequently Asked Questions About Dynamic Invoke

This section includes a collection of questions that you might encounter when you are dynamically invoking methods to test custom controls.

Which Methods Can I Call With the `invoke` Method?

To get a list of all the methods that you can call with the `invoke` method for a specific test object, you can use the `getDynamicMethodList`. To view the list, you can for example print it to the console or view it in the debugger.

Why Does an Invoke Call Return a Simple String when the Expected Return is a Complex Object?

The `invoke` method can only return simple data types. Complex types are returned as string. Silk4J uses the `ToString` method to retrieve the string representation of the return value. To call the individual methods and read properties of the complex object that is returned by the first method invocation, use `invokeMethods` instead of `invoke`.

How Can I Simplify My Scripts When I Use Many Calls To `invokeMethods`?

When you extensively use `invokeMethods` in your scripts, the scripts might become complex because you have to pass all method names as strings and all parameters as lists. To simplify such complex scripts, create a static method that interacts with the actual control in the AUT instead of interacting with the control through `invokeMethods`. For additional information, see *Adding Code to the Application Under Test to Test Custom Controls*.

Adding Code to the Application Under Test to Test Custom Controls

When you are testing Windows Forms applications or WPF applications, and you want to test complex custom controls or custom controls that you cannot test by simply using the `invoke` and `invokeMethods` methods, you can create a static method that interacts with the actual control in the application under test (AUT) and you can add this code to the AUT.

The benefit for you from adding code to the AUT is that the code in the AUT can use regular method calls for interacting with the control, instead of using the reflection-like style of calling methods with the dynamic invoke methods. Therefore you can use code completion and IntelliSense when you are writing your code. You can then call the code in the AUT with a simple `invoke` call, where you pass the control of interest as a parameter.

You can add code to the AUT in the following ways:

- Compile the code into the AUT. The implementation is simple, but you will be changing the AUT, which you might not want to do.
- Inject code to the AUT at runtime by using the `LoadAssembly` method in a test script. This requires more effort than compiling the code into the AUT, but the injected code will be located close to the test code. The `LoadAssembly` method is available for the classes `WPFWindow` and `FormsWindow`.

Example: Testing the UltraGrid Infragistics control

This example demonstrates how you can retrieve the content of an UltraGrid control. The UltraGrid control is included in the NETAdvantage for Windows Forms library which is provided by Infragistics. You can download a trial of the library from <http://www.infragistics.com/products/windows-forms/downloads>.

To create the UltraGridUtil class, perform the following actions:

1. Open Microsoft Visual Studio and create a new class library project in C# or VB .NET. Call the new project AUTEExtensions.



Note: The class library should use the same .NET version as the AUT.

2. Add references to the required dependencies to the project. For example, for Infragistics version 12.2 you need to reference the following assemblies:
 - Infragistics4.Shared.v12.2
 - Infragistics4.Win.UltraWinGrid.v12.2
 - Infragistics4.Win.v12.2

If you are not sure which version of Infragistics is used in your AUT you can use the **Process Explorer** tool from Microsoft to see which assemblies are loaded in your AUT.

- a. In the AUTEExtensions project, create the new class UltraGridUtil with the following content:

```
' VB code
Public Class UltraGridUtil

    Public Shared Function GetContents(ultraGrid As
Infragistics.Win.UltraWinGrid.UltraGrid) As List(Of List(Of
String))
    Dim contents = New List(Of List(Of String))
    For Each row In ultraGrid.Rows
        Dim rowContents = New List(Of String)
        For Each cell In row.Cells
            rowContents.Add(cell.Text)
        Next
        contents.Add(rowContents)
    Next
    Return contents
End Function

End Class
```

```
// C# code
using System.Collections.Generic;

namespace AUTEExtensions {

    public class UltraGridUtil {

        public static List<List<string>>
GetContents(Infragistics.Win.UltraWinGrid.UltraGrid grid) {
            var result = new List<List<string>>();
            foreach (var row in grid.Rows) {
                var rowContent = new List<string>();
                foreach (var cell in row.Cells) {
                    rowContent.Add(cell.Text);
                }
                result.Add(rowContent);
            }
        }
    }
}
```

```
        return result;
    }
}
}
```



Note: The `Shared` modifier makes the `GetContents` method a static method.

3. Build the `AUTExtensions` project.
4. Load the assembly into the AUT during playback.
 - Open an existing test script or create a new test script.
 - Add code to the test script to load the assembly that you have built from the file system. For example:
5. Call the static method of the injected code in order to get the contents of the `UltraGrid`:

```
// Java code
Control ultraGrid = mainWindow.find("//
Control[@automationId='my grid']");
List<List<String>> contents = (List<List<String>>)
mainWindow.invoke("AUTExtensions.UltraGridUtil.GetContents",
ultraGrid);
```

Frequently Asked Questions About Adding Code to the AUT

This section includes a collection of questions that you might encounter when you are adding code to the AUT to test custom controls.

Why is Code That I Have Injected Into the AUT With the `LoadAssembly` Method Not Updated in the AUT?

If code in the AUT is not replaced by code that you have injected with the `LoadAssembly` method into the AUT, the assembly might already be loaded in your AUT. Assemblies cannot be unloaded, so you have to close and re-start your AUT.

Why Do the Input Argument Types Not Match When I Invoke a Method?

If you invoke a method and you get an error that says that the input argument types do not match, the method that you want to invoke was found but the arguments are not correct. Make sure that you use the correct data types in your script.

If you use the `LoadAssembly` method in your script to load an assembly into the AUT, another reason for this error might be that your assembly is built against a different version of the third-party library than the version that is used by the AUT. To fix this problem, change the referenced assembly in your project. If you are not sure which version of the third-party library is used in your AUT, you can use the **Process Explorer** tool from Microsoft.

How Do I Fix the Compile Error when an Assembly Can Not Be Copied?

When you have tried to add code to the AUT with the `LoadAssembly` method, you might get the following compile error:

Could not copy '<assembly_name>.dll' to '<assembly_name>.dll'. The process cannot access the file. The reason for this compile error is that the assembly is already loaded in the AUT and cannot be overwritten.

To fix this compile error, close the AUT and compile your script again.

Testing Apache Flex Custom Controls

Silk4J supports testing Apache Flex custom controls. However, by default, Silk4J cannot record and playback the individual sub-controls 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 Apache 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 Apache Flex control.
- Seamless integration into Silk4J API
- Silk4J uses the specific class name in the locator, for example Silk4J records `//FlexSpinner`.

The disadvantages of advanced support include:

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

Managing Custom Controls

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

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

Example: Testing the UltraGrid Infragistics control

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

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

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

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

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

Typically, you can implement the methods in a custom control class in one of the following ways:

- You can use methods like `click`, `typeKeys`, `textClick`, and `textCapture`.
- You can dynamically invoke methods on the object in the AUT.
- You can dynamically invoke methods that you have added to the AUT. This is the approach that is described in this example.

You can use the following code to call the static method that is defined in the example in *Adding Code to the Application Under Test to Test Custom Controls*. The method `GetContents` is added into the generated class `UltraGrid`.

```
// Java code
import
com.borland.silktest.jtf.Desktop;
import
com.borland.silktest.jtf.common.JtfObjectHandle;

public class UltraGrid extends
com.borland.silktest.jtf.Control {

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

    public List<List<String>>
    getContents() {
        return (List<List<String>>)
        invoke("AUTExtensions.UltraGridUtil.
        GetContents", this);
    }
}
```

When you define a class as a custom control, you can use the class in the same way in which you

can use any built-in class, for example the Dialog class.

```
// Java code
UltraGrid ultraGrid =
mainWindow.find("//
UltraGrid[@automationId='my
grid']");
List<List<String>> contents =
ultraGrid.getContents();
```

Supporting a Custom Control

Silk4J supports managing custom controls over the UI for the following technology domains:

- Win32
- Windows Presentation Foundation (WPF)
- Windows Forms
- Java AWT/Swing
- Java SWT

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

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

A new row is added to the list of custom controls.

6. If you have chosen to manually add a custom control to the list:
 - a) In the **Silk Test base class** column, select an existing base class from which your class will derive. This class should be the closest match to your type of custom control.
 - b) In the **Silk Test class** column, enter the name to use to refer to the class. This is what will be seen in locators. For example: `//UltraGrid` instead of `//Control[13]`.
7. *Only for Win32 applications:* In the **Use class declaration** column, set the value to **False** to simply map the name of a custom control class to the name of a standard Silk Test class.



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

- c) In the **Custom control class name** column, enter the fully qualified class name of the class that is being mapped. For example: `Infragistics.Win.UltraWinGrid.UltraGrid`. For Win32 applications, you can use the wildcards `?` and `*` in the class name.
7. *Only for Win32 applications:* In the **Use class declaration** column, set the value to **False** to simply 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. Set the value to **True** to additionally use the class declaration of the custom control class.
 8. Click **OK**.
 9. *Only for scripts:*

- a) Add custom methods and properties to your class for the custom control.
- b) Use the custom methods and properties of your new class in your script.

 **Note:** The custom methods and properties are not recorded.

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

Custom Controls Options

Silk4J > Manage Custom Controls.

Silk4J supports managing custom controls over the UI for the following technology domains:

- Win32
- Windows Presentation Foundation (WPF)
- Windows Forms
- Java AWT/Swing
- Java SWT

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

When you map a custom control class to a standard Silk Test class, you can use the functionality supported for the standard Silk Test class in your test. The following **Custom Controls** options are available:

Option	Description
Silk Test base class	Select an existing base class to use that your class will derive from. This class should be the closest match to your type of custom control.
Silk Test class	Enter the name to use to refer to the class. This is what will be seen in locators.
Custom control class name	Enter the fully qualified class name of the class that is being mapped. You can use the wildcards ? and * in the class name.
Use class declaration	This option is available only for Win32 applications. By default <code>False</code> , which means the name of the custom control class is mapped to the name of the standard Silk Test class. Set this setting to <code>True</code> to additionally use the class declaration of the custom control class.

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

Example: Setting the options for the UltraGrid Infragistics control

To support the `UltraGrid Infragistics` control, use the following values:

Option	Value
Silk Test base class	<code>Control</code>
Silk Test class	<code>UltraGrid</code>
Custom control class name	<code>Infragistics.Win.UltraWinGrid.UltraGrid</code>

Improving Object Recognition with Microsoft Accessibility

You can use Microsoft Accessibility (Accessibility) to ease the recognition of objects at the class level. There are several objects in Internet Explorer and in Microsoft applications that Silk4J can better recognize if you enable Accessibility. For example, without enabling Accessibility Silk4J records only basic information about the menu bar in Microsoft Word and the tabs that appear. However, with Accessibility enabled, Silk4J fully recognizes those objects.

Example

Without using Accessibility, Silk4J cannot fully recognize a `DirectUIHwnd` control, because there is no public information about this control. Internet Explorer uses two `DirectUIHwnd` controls, one of which is a popup at the bottom of the browser window. This popup usually shows the following:

- The dialog box asking if you want to make Internet Explorer your default browser.
- The download options **Open**, **Save**, and **Cancel**.

When you start a project in Silk4J and record locators against the `DirectUIHwnd` popup, with accessibility disabled, you will see only a single control. If you enable Accessibility you will get full recognition of the `DirectUIHwnd` control.

Using Accessibility

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

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

Example

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

- Control
 - Control
- Button

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

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

Enabling Accessibility

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

To enable Accessibility:

1. Click **Silk4J > Edit Options**. The **Script Options** dialog box opens.
2. Click **Advanced**.
3. Select the **Use Microsoft Accessibility** option. Accessibility is turned on.

Overview of Silk4J Support of Unicode Content

The Open Agent is Unicode-enabled, which means that the Open Agent is able to recognize double-byte (wide) languages.

With Silk4J you can test applications that contain content in double-byte languages such as Chinese, Korean, or Japanese (Kanji) characters, or any combination of these.

The Open Agent supports the following:

- Localized versions of Windows.
- International keyboards and native language Input Method Editors (IME).
- Passing international strings as parameters to test cases, methods, and so on, and comparing strings.
- Reading and writing text files in multiple formats: ANSI, Unicode, and UTF-8.

For information about new features, supported platforms, and tested versions, refer to the [Release Notes](#).

Before testing double-byte characters with Silk4J

Testing an internationalized application, particularly one that contains double-byte characters, is more complicated than testing an application that contains strictly English single-byte characters. Testing an internationalized application requires that you understand a variety of issues, from operating system support, to language packs, to fonts, to working with IMEs and complex languages.

Before you begin testing your application using Silk4J, you must do the following:

- Meet the needs of your application under test (AUT) for any necessary localized OS, regional settings, and required language packs.
- Install the fonts necessary to display your AUT.
- If you are testing an application that requires an IME for data input, install the appropriate IME.

Microsoft UI Automation

Microsoft UI Automation (UI Automation) is a framework that enables you to access, identify, and manipulate UI elements of any application by providing programmatic access to these user interface elements. When testing against Windows-based applications that have implemented UI Automation provider interfaces, you can use UI Automation to improve the object recognition for the controls in these applications. In this Help, we will refer to such controls as *UI Automation controls*.



Note: Silk4J supports testing Windows-based applications that have implemented UI Automation on machines with Microsoft Windows 8 or later.

UI Automation provides fallback support for applications that are based on the following technologies:

- Win32
- WPF
- WinForms
- Oracle JavaFX
- QT
- PowerBuilder

- Delphi
- Microsoft Office

For example, if you cannot record a test against your application because Silk4J cannot recognize the objects in the application or because Silk4J recognizes all objects in the application as `Control`, you could try to enable the UI Automation support.

To enable the UI Automation support during recording, stop recording, enable the option **OPT_ENABLE_UI_AUTOMATION_SUPPORT**, and resume recording. For additional information, see *Setting UI Automation Options*.

 **Note:** The UI Automation support overrides the standard technology-domain-specific support. When you are finished interacting with the controls that require UI Automation support, disable the UI Automation support again to resume working with standard controls.

 **Note:** If you are testing against a Java FX application, you do not have to enable the UI Automation support, as Silk4J enables this out-of-the-box for Java FX applications.

To ensure that the methods supported for a UI Automation control cover the corresponding controls in all supported technologies, the Silk4J API supports only a subset of the methods and properties available for these controls. To call additional methods and properties that are not available in the Silk4J API for a control, use dynamic invoke.

Recording a Test Against an Application with an Implemented UI Automation Provider Interface

Use the **Interactive Recording** window to record a test against a windows-based application which has implemented UI Automation provider interfaces.

 **Note:** Silk4J supports testing Windows-based applications that have implemented UI Automation on machines with Microsoft Windows 8 or later.

Before you can record a Silk4J test, you must have created a Silk4J project.

To record a new test for a windows-based application which has implemented UI Automation provider interfaces:

1. Select the project to which you want to add the new test.
2. In the toolbar, click **Record Actions**.
3. The **Interactive Recording** window opens and displays the application under test. Perform the actions that you want to record.
 - a) Click on the object with which you want to interact. Silk4J performs the default action for the object. If there is no default action, or if you have to insert text or specify parameters, the **Choose Action** dialog box opens.
 - b) *Optional:* To choose an action for an object, which might not be the default action, right-click on the object. The **Choose Action** dialog box opens.
 - c) *Optional:* If the action has parameters, type the parameters into the parameter fields. Silk4J automatically validates the parameters.
 - d) Click **OK** to close the **Choose Action** dialog box. Silk4J adds the action to the recorded actions and replays it on the mobile device or emulator.

During recording, Silk4J displays the mouse position next to the recording window. You can toggle the location to switch between displaying the absolute mouse position on the device display and the mouse position in relation to the active object. For additional information about the actions available during recording, see *Actions Available During Recording*.

4. Click **Stop**. The **Record Complete** dialog box opens.
5. The **Source folder** field is automatically populated with the source file location for the project that you selected. To use a different source folder, click **Select** and navigate to the folder that you want to use.

6. *Optional:* In the **Package** text box, specify the package name.
For example, type: `com.example`.
To use an existing package, click **Select** and select the package that you want to use.
7. In the **Test class** text box, specify the name for the test class.
To use an existing class, click **Select** and select the class that you want to use.
8. In the **Test method** text box, specify a name for the test method.
9. Click **OK**.

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

Dynamically Invoking UI Automation Methods

To ensure that the methods supported for a UI Automation control cover the corresponding controls in all supported technologies, the Silk4J API supports only a subset of the methods and properties available for these controls. To call additional methods and properties that are not available in the Silk4J API for a control, use 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.

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.Visibility` 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

This example shows how you can call the scrolling methods of a `UIADocument` control by using dynamic invoke. Silk4J does not expose these scrolling methods in the API, as these methods are not available for the `UIADocument` control in all technologies that have implemented UI Automation provider interfaces.

To see which methods and properties are available for the control, you could use code similar to the following:

```
UIADocument textBox = mainWindow.<UIADocument>find("//
UIADocument");
List<String>propertyList = textBox.getPropertyList();
List<String>methodList = textBox.getDynamicMethodList();
```

For this example, the *propertyList* that is returned by the `GetPropertyList` method includes the property `ScrollPattern.VerticalScrollPercent`. The *methodList* that is returned by the `GetDynamicMethodList` method includes the method `ScrollPattern.ScrollVertical`.

By using dynamic invoke, you can call the method `ScrollPattern.ScrollVertical` as follows:

```
textBox.invoke("ScrollPattern.ScrollVertical",
ScrollAmount.SMALL_INCREMENT);
```

Alternatively, you can call the property `ScrollPattern.VerticalScrollPercent` as follows:

```
textBox.getProperty("ScrollPattern.VerticalScrollPercent");
```

Locator Attributes for Identifying Controls with UI Automation

The supported locator attributes for controls in Windows-based applications that have implemented UI Automation provider interfaces include:

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



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

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

To identify controls in Windows-based applications that have implemented UI Automation provider interfaces within 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 `//UIAButton[@automationId="okButton"]`.

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

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

During recording, Silk4J creates a locator for a UI Automation 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.

To find out which additional custom attributes you could use for the UI Automation controls in your AUT, you can use the **Verify Properties** dialog box. To do so, hover the mouse cursor over a UI Automation control during recording, and click **Ctrl+Alt**. You can then see which properties are available for the control. For example, for some applications, the attribute `value` is useful.

Scrolling in UI Automation Controls

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

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

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

Controls that support scrolling

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

Examples

The following command scrolls a list box to the bottom:

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

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

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

Controls that do not support scrolling

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

Examples

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

```
textBox.UIAVerticalScrollBar().ScrollToPosition(15)
```

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

```
textBox.UIAVerticalScrollBar().ScrollToMaximum()
```

Limitations when Using UI Automation

The known limitations when using UI Automation are:

No support for IMEs while using UI Automation support

While the UI Automation support is enabled, Silk4J provides no support for using Input Method Editors (IMEs).

Troubleshooting when Testing with UI Automation Support Enabled

Why does a script with UI Automation controls that is recorded on Microsoft Windows 7 not replay on Microsoft Windows 8 or later?

When you record a script that includes UI Automation controls on Microsoft Windows 7 or prior, and then try to replay it on Microsoft Windows 8 or later, the replay might fail. That is because Microsoft has changed the underlying automation, and the UI Automation behave differently between those Windows versions.

For example, some UI Automation controls in an application might have a value for the `automationId` attribute on Microsoft Windows 7 and no value for the same attribute on Microsoft Windows 10.

In such a case, Micro Focus recommends recording the script again against the later Microsoft Windows version.

Why is the first action in a Microsoft Office application not replayed?

If you are recording against a Microsoft Office application, the application window needs to be active before recording so that Silk4J can replay all actions correctly. For example, if you are recording against Microsoft Excel, and you have changed the focus to another application, the first click in Excel that is recorded by Silk4J will not do anything during replay. You will have to repeat the recorded action and remove it from your test.

As a workaround, left-click into the title bar of the Microsoft Office application before recording any actions. This click into the title bar is not recorded by Silk4J, and you can continue recording as intended.

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	<input checked="" type="checkbox"/>	MasterCard
Brian Miller	02.01.2008	1	<input type="checkbox"/>	Visa
Caral Rudd	03.01.2008	2	<input checked="" type="checkbox"/>	American Ex...
Dan Rundgren	04.01.2008	3	<input type="checkbox"/>	MasterCard
Devie Yingstein	05.01.2008	4	<input checked="" type="checkbox"/>	Visa

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

```
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,

which do not support Direct3D, 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
```

Text recognition is not supported for Java Applets and Swing applications that support Direct3D.

- Internet Explorer.
- WebDriver-based browsers.



Note: Text recognition does not work with controls that are not visible on the screen. For example, you cannot use text recognition for a text that is scrolled out of view.



Note: Text recognition might not work if the font that is used in the target text is not installed on the machine on which the test is executed.

WebDriver-based browsers

The text recognition methods can be applied to `BrowserWindow` and `DomElement` objects.



Note: Text recognition does not work for text that is drawn in `<canvas>` elements.



Note: Text recognition does not work for content added by CSS pseudo-elements like `::before` and `::after`.

Text recognition methods

Silk4J offers the following methods to drive testing through interacting with the text that the AUT renders on the screen:

- | | |
|----------------------|---|
| TextCapture | Returns the text that is within a control. Also returns text from child controls. |
| TextClick | Clicks on a specified text within a control. Waits until the text is found or the <i>Object resolve timeout</i> , which you can define in the synchronization options, is over. |
| TextRectangle | Returns the rectangle of a certain text within a control or a region of a control. |
| TextExists | Determines whether a given text exists within a control or a region of a control. |

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.

The methods `TextClick`, `TextRectangle`, and `TextExists` internally use `TextCapture` to grab the visible text from the application and allow for further processing of that text. The underlying `TextCapture` method is implemented in two different ways. Silk4J decides which implementation to use depending on the type of the application under test.

- For native windows applications, including WPF, WinForms, and Java applications, but also Internet Explorer, Silk4J hooks into the text rendering functions of the Windows API to extract the text that the application draws on the screen.
- For Google Chrome, Mozilla Firefox, Microsoft Edge, and Apple Safari, Silk4J uses a JavaScript-based approach to retrieve the text after it was rendered by the browser.



Note: Because of the different nature of these two implementations, Silk4J might return different text for the same web application, depending on which browser is used.

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);
```

Service Virtualization

Using *service virtualization* (SV) with Silk4J enables you to simulate back-end services of a web application. This allows you to easily test components of complex web applications which you cannot properly access or configure for testing. When running tests in Silk4J, you can use a virtualized service with your test in place of a real service of your application under test (AUT).

For example, using a real credit card service when testing an AUT that provides the ability to purchase something is not practical, as a real credit card will need to be charged for each purchase. In such a case, using a virtualized service during testing is preferable.

Silk4J utilizes service virtualization functionality provided by SV Lab. For detailed information about the capabilities of SV Lab, refer to the [SV Lab documentation](#).

Service virtualization in Silk4J includes the following stages:

- Creating a new Silk4J test for your application.
- Configuring the browsers against which you want to test your application to use the service virtualization proxy server.
- Discovering the endpoints in your application.
- Learning which data is transmitted through specific endpoints.
- Integrating the service virtualization with the test.
- Customizing the virtualization script.
- Simulating the web service behavior by transmitting simulated data through the endpoints.



Tip: In addition to the capabilities provided in the Silk4J UI, you can use the `svlab.properties` file to configure some settings for service virtualization, for example the outbound proxy server that is used to connect to the web. The file is located in the Silk Test application data directory, under `%APPDATA%/Silk/SilkTest/conf/`.

Discovering the Endpoints in your Application Under Test

Use the **DISCOVERY** tab of the **SERVICE VIRTUALIZATION** user interface to discover the endpoints in a web application.

1. Click the drop-down arrow next to the Silk Test toolbar icon  and choose **Service Virtualization**. The **Service Virtualization** user interface appears.
2. Select the **DISCOVERY** tab.
3. Click **DISCOVER**. Silk4J starts the proxy that is displayed in the **Proxy Address** field.
4. Open the browser against which you want to test the AUT. For example, open Mozilla Firefox.

5. Configure internet access through the proxy that is displayed in the **Proxy Address** field of the **Service Virtualization** user interface for the browser.
For example, configure `http://localhost:9000` as the proxy for Mozilla Firefox.
For information on configuring a proxy for a specific browser, refer to the documentation of the browser.
6. Ensure that the browser has not cached any data about the AUT.
7. Type the address of the AUT into the address bar of the browser.
For example, to discover the endpoints in the demo application that is provided by Micro Focus, type <http://www.advantageonlineshopping.com>.
8. When the selected page is fully displayed in the browser, click **STOP** in the **SERVICE VIRTUALIZATION** user interface. Silk4J displays the discovered endpoints in the **Endpoints** field.
9. Check the corresponding check boxes in the **Endpoints** field to select up to five of the discovered endpoints.
For example, to select the POPULAR ITEMS page in the demo application as an endpoint, check the check box before `http://www.advantageonlineshopping.com/app/tempFiles/popularProducts.json`.
10. Click **LEARN** to continue learning which data is transmitted through the selected endpoints.
For additional information, see [Learning which Data is Transmitted Through an Endpoint](#).

Learning which Data is Transmitted Through an Endpoint

Use the **LEARNING** tab of the **SERVICE VIRTUALIZATION** user interface to learn which data is transmitted through selected endpoints in a web application.

Before you can learn which data is transmitted through endpoints, discover the endpoints. For additional information, see [Discovering the Endpoints in your Application Under Test](#).

1. Click the drop-down arrow next to the Silk Test toolbar icon  and choose **Service Virtualization**. The **Service Virtualization** user interface appears.
2. Select the **LEARNING** tab.
3. Type a name for the new scenario into the **Scenario Name** field.
For example, type `ChangePopularProductName`.
Silk4J saves new scenarios in the `Scenarios` subfolder of the project.



Tip: In addition to the capabilities provided in the Silk4J UI, you can use the `svlab.properties` file to configure some settings for service virtualization, for example the outbound proxy server that is used to connect to the web. The file is located in the Silk Test application data directory, under `%APPDATA%/Silk/SilkTest/conf/`.

4. Select at least one endpoint from the **Endpoints** field or manually add an endpoint by typing it into the **Add Endpoint** field.
5. Click **LEARN**. Silk4J starts the proxy that is displayed in the **PROXY URL** field.
6. Open the browser against which you want to test the AUT.
For example, open Mozilla Firefox.
7. Ensure that the proxy that is displayed in the **Proxy URL** field is configured for the browser.
8. Ensure that the browser has not cached any data about the AUT.
9. Perform some actions in the AUT to ensure data is saved to the scenario.
For example, click **POPULAR ITEMS** in the demo application (<http://www.advantageonlineshopping.com>).
10. Click **SIMULATE** to continue simulating the data that is transmitted through the selected endpoints.
For additional information, see [Simulating Data for Selected Endpoints](#).

Simulating Data for Selected Endpoints

Use the **SIMULATE** tab of the **SERVICE VIRTUALIZATION** user interface to simulate data that is transmitted through selected endpoints in a web application.

Before you can simulate data that is transmitted through an endpoint, learn which data is transmitted through the endpoint. For additional information, see [Learning which Data is Transmitted Through an Endpoint](#).

 **Note:** With Silk4J, you can simulate up to three endpoints at the same time. If you require additional endpoints, you have to use an SV Lab license. You can specify a license server for SV Lab by setting the `SVLicenseServerUrl` property in the `svlab.properties` file. The file is located in the Silk Test application data directory, under `%APPDATA%/Silk/SilkTest/conf/`.

1. Click the drop-down arrow next to the Silk Test toolbar icon  and choose **Service Virtualization**. The **Service Virtualization** user interface appears.
2. Select the **SIMULATION** tab.
3. Change the data for the endpoint in the JavaScript file of the endpoint to the new data that you want to simulate.
Silk4J saves new scenarios in the `Scenarios` subfolder of the project. The JavaScript file has the suffix `jsonModel` and is located in a subfolder of the `Scenarios` folder with the name of the scenario.
4. Select the scenario from the **Scenarios** list.
5. Click **SIMULATE**.
You can specify the proxy port that is used while simulating the data in the file `sv-lab.json`, which is located in the scenario folder. By default, this is the same port as the one that is used for learning.
6. Ensure that the browser has not cached any data about the AUT.
7. Perform an action in the AUT that will cause the changed data to be transmitted through the endpoint.
8. If the simulation was successful, click **STOP**.

Simulating Data for Selected Endpoints through the API

Use the API to simulate data that is transmitted through selected endpoints in a web application.

Before you can simulate data that is transmitted through an endpoint, learn which data is transmitted through the endpoint. For additional information, see [Learning which Data is Transmitted Through an Endpoint](#).

 **Note:** With Silk4J, you can simulate up to three endpoints at the same time. If you require additional endpoints, you have to use an SV Lab license. You can specify a license server for SV Lab by setting the `SVLicenseServerUrl` property in the `svlab.properties` file. The file is located in the Silk Test application data directory, under `%APPDATA%/Silk/SilkTest/conf/`.

To add service virtualization functionality to an existing Silk4J script:

1. Add the *SV Lab Client Library* to the project that contains the Silk4J script.
 - a) In the **Package Explorer**, right-click on the project node.
 - b) Select **Silk4J Tools > Add Service Virtualization Capability**.
2. Change the data for the endpoint in the JavaScript file of the endpoint to the new data that you want to simulate.
Silk4J saves new scenarios in the `Scenarios` subfolder of the project. The JavaScript file has the suffix `jsonModel` and is located in a subfolder of the `Scenarios` folder with the name of the scenario.
3. Open the Silk4J script.

4. Change the desktop to static.

```
private static Desktop desktop = new Desktop();
```

5. Add a SvClient member variable.

```
private static SvClient sv;
```

6. Start the SV Lab and use the simulated data.

a) Add the following imports to the start of the script.

```
import org.junit.BeforeClass;
import com.microfocus.silktest.jtf.sv.SvLabIntegration;
import org.microfocus.sv.api.SvClient;
import org.microfocus.sv.model.project.Module;
```

b) Add the following code before the test to simulate the endpoint data.

```
@BeforeClass
public static void startSVLabSimulation() {
    String scenarioName = "PopularProducts";
    SvLabIntegration svLabIntegration = desktop.getSvLabIntegration();
    String endpoint = svLabIntegration.getServerEndpoint();
    SvClient sv = SvClient.newInstance(endpoint);
    Module module = sv.compileModuleFromSources("classpath:/" +
scenarioName + "/*");
    sv.loadActiveVirtualLab("classpath:/" + scenarioName + "/sv-lab.json",
module, true);
    sv.startActiveVirtualLab();
    sv.runSimulation(scenarioName);
}
```



Note: The *scenarioName* is the name of a scenario in the current project.

7. Ensure that the proxy is specified in the base state.

8. Ensure that the browser has not cached any data about the AUT.

9. Run the test script.

10. Stop the SV Lab.

a) Add the following imports to the start of the script.

```
import org.junit.AfterClass;
```

b) Add the following code after the test.

```
@AfterClass
public static void stopSVLab() {
    if (sv != null) {
        sv.close();
    }
}
```

11. Replay the test.

For information about replaying tests from the command line, see *Replaying a Test from the Command Line*.

Example: Using simulated data in a test

The entire script that simulates the data for the endpoint should look similar to the following.

```
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.microfocus.sv.api.SvClient;
import org.microfocus.sv.model.project.Module;

import com.borland.silktest.jtf.BrowserBaseState;
```

```

import com.borland.silktest.jtf.Desktop;
import com.borland.silktest.jtf.xbrowser.DomElement;
import com.microfocus.silktest.jtf.sv.SvLabIntegration;

public class MyTest {

    private static Desktop desktop = new Desktop();
    private static SvClient sv;

    @BeforeClass
    public static void startSVLabSimulation() {
        String scenarioName = "PopularProducts";
        SvLabIntegration svLabIntegration =
desktop.getSvLabIntegration();
        String endpoint = svLabIntegration.getServerEndpoint();
        sv = SvClient.newInstance(endpoint);
        Module module = sv.compileModuleFromSources("classpath:/" +
scenarioName + "/*");
        sv.loadActiveVirtualLab("classpath:/" + scenarioName + "/sv-
lab.json", module, true);
        sv.startActiveVirtualLab();
        sv.runSimulation(scenarioName);
    }

    @Before
    public void baseState() {
        // Go to web page 'http://advantageonlineshopping.com'
        BrowserBaseState baseState = new BrowserBaseState();
        baseState.execute(desktop);
    }

    @Test
    public void testSimulatedProduct() {
        DomElement firstPopularItem = desktop.<DomElement> find("//
BrowserApplication//BrowserWindow//
P[@name='popular_item_16_name']");
        Assert.assertEquals("SIMULATED: HP ELITEPAD 1000 G2
TABLET", firstPopularItem.getText());
    }

    @AfterClass
    public static void stopSVLab() {
        if (sv != null) {
            sv.close();
        }
    }
}

```

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. For additional information, see [Grouping tests using JUnit categories](#).

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 example shows how you can execute the `Silk4J` tests that are included in a category.

To import the `Category` class you will need to add a line similar to the following to the start of your test script:

```
import org.junit.experimental.categories.Category;
```

Categories can be implemented as classes or as interfaces, for example:

```
public interface FastTests {}
public interface SlowTests {}
```

You can flag an entire class with a category. In the following code sample, all methods in the class are flagged with the category `SlowTests`:

```
@Category( { SlowTests.class })
public class A {
    @Test
    public void a() {
        ...
    }

    @Test
    public void b() {
        ...
    }
}
```

You can also flag individual methods in a class with a category. In the following code sample, only the method `d` is flagged with the category `FastTests`:

```
public class B {
    @Test
    public void c() {
        ...
    }

    @Category(FastTests.class)
    @Test
    public void d() {
        ...
    }
}
```

You can flag a class or method with multiple categories:

```
@Category( { SlowTests.class, FastTests.class })
public static class C {
    @Test
    public void e() {
        ...
    }
}
```

To run tests in a particular category, you need to set up a test suite:

```
@RunWith(SilkTestCategories.class)
@IncludeCategory(SlowTests.class)
@SuiteClasses( { A.class, C.class })
```

```
// Note: SilkTestCategories is a kind of Suite
public static class SlowTestSuite {}
```

Why Do I Get the Error: Category cannot be resolved to a type?

If you want to use categories to group Silk4J tests, and you are faced with the error `Category cannot be resolved to a type`, your test class does probably not import the `Category` class.

To import the `Category` class you will need to add a line similar to the following to the start of your test script:

```
import org.junit.experimental.categories.Category;
```

Inserting a Result Comment in a Script

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

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

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

Consuming Parameters from Silk Central

To enable Silk4J to use a parameter that has been set for a test in Silk Central, use the method `System.getProperty("myparam")`.

Configuration Testing with Silk Central Connect

To work with Silk Central, ensure that you have configured a valid Silk Central location. For additional information, see [Integrating Silk4J with Silk Central](#)

To execute your automated tests on a variety of *configurations*, which are combinations of operating systems and Web browsers, you can use Silk Central Connect. Silk Central Connect is a tool that combines aspects of test execution management and configuration testing into an easy to use interface, providing the following advantages:

- Simple execution of all your automated unit tests on a variety of configurations.
- Leverages the advantages of the Amazon Web Services, enabling you to easily access a variety of configurations without any upfront investment.
- Tight integration between Silk Central Connect and Silk4J for easy test creation, maintenance, and execution.
- Side-by-side result analysis, enabling you to see how all of your tests look like across the different configurations.

For additional information about Silk Central Connect, refer to the [Silk Central Connect Help](#).

For information about installing, deploying, and licensing Silk Central Connect, refer to the [Silk Central Installation Help](#).

For information about configuring your test environment, see [Setting Up Execution Servers](#).

Measuring Execution Time

You can use methods and properties provided by the `Timer` class to measure the time that your tests require to execute. For additional information, see *Timer Class* in the Javadoc.

Among other usages, these methods and properties are used for the timing of test executions that are triggered from Silk Performer. For additional information on integrating Silk4J with Silk Performer, refer to the [Silk Performer Help](#).

Slowing Down Tests

Some applications under test might require extensive loading of application data in the UI, and might not be finished on time with loading objects that are required for replaying a test. To successfully replay tests on such an AUT, you can check for the existence of an object before performing an action on it, or you can add sleeps before performing an action.



Note: Micro Focus does not recommend generally adding sleeps to tests, because in most cases Silk4J will automatically detect if an object is available, and sleeps might severely reduce the performance of tests.

1. To check if an object is available in the AUT, use the `exists` method.

For example, to wait for six seconds for the button `INPUT` to become available, add the following line to your test script:

```
browserWindow.exists("//INPUT", 6000);
```

2. To add a sleep before performing an action on a control, use the `sleep` method.

For example, to sleep for six seconds, add the following line to your test script:

```
Utils.sleep(6000);
```

For additional information on these methods, see the Javadoc.

Testing Applications in Multiple UI Sessions on a Single Machine

To test applications in multiple UI sessions on a single machine or to test multiple agents on a single machine, connect to multiple Open Agent instances on the machine. Every agent runs in its own UI-session. A UI session can be a Remote Desktop Protocol (RDP) connection or a Citrix-based connection.

1. Create the UI sessions.
2. Open a command line window.
3. Navigate to the folder `/ng/agent` in the Silk Test installation directory.

For example, the default folder path might look like the following: `C:\Program Files (x86)\Silk\SilkTest\ng\agent`.

4. In each UI session, execute the following command: `openAgent.exe -infoServicePort=<port>`.



Note: Use a unique port number, because this port will be used in your Silk4J script to identify the Open Agent and the UI session in which the agent is running.

5. Change your Silk4J scripts to connect to the Open Agent instances.

To connect to an Open Agent instance, add the following line to the script:

```
Desktop desktopSession = new Desktop("hostname:port");
```

Where *hostname* is the name of the machine on which the agent is running, and *port* is the unique port that you have specified.

The resulting objects are independent of each other and can be used either in one thread or in multiple threads.

 **Note:** If you want to launch an application in multiple UI sessions, you have to execute the base state for each UI session.

 **Note:** To use TrueLog when testing applications in multiple UI sessions on a remote machine, you need to manually copy any generated TrueLog files from the remote machine to your local machine.

Example

Assume that the server machine that is hosting the UI sessions is named *ui-srv*. You can create three UI sessions by using the ports 22903, 22904, and 22905.

In the first session, open the command line window, navigate to the `agent` directory, and type the following:

```
openAgent.exe -infoServicePort=22903
```

Do the same for the other two sessions with the respective ports 22904 and 22905.

To connect to the Open Agent instances, add the following code to your script:

```
Desktop desktopSession1 = new Desktop("ui-srv:22903");  
Desktop desktopSession2 = new Desktop("ui-srv:22904");  
Desktop desktopSession3 = new Desktop("ui-srv:22905");
```

The following sample script prints a simple text to each of the three UI sessions:

```
public class TestMultiSession {  
    Desktop d1 = new Desktop("ui-srv:22903");  
    Desktop d2 = new Desktop("ui-srv:22904");  
    Desktop d3 = new Desktop("ui-srv:22905");  
  
    @Test  
    public void test() {  
        BaseState basestate = new BaseState();  
        basestate.execute(d1);  
        basestate.execute(d2);  
        basestate.execute(d3);  
  
        d1.<Window>find("//Window").typeKeys("Hello to session 1!");  
        d2.<Window>find("//Window").typeKeys("Hello to session 2!");  
        d3.<Window>find("//Window").typeKeys("Hello to session 3!");  
    }  
}
```

Encrypting Passwords

Encrypt password sensitive strings that are plain text to avoid security issues.

1. In the menu, select **Silk4J > Password Encoder**. The **Password Encoder** dialog appears.
2. Type the string to be encrypted into the **Password** field. The encrypted password is displayed in the **Encoded string** field.
3. Click **Copy and Close** to copy the encrypted password to the clipboard and to close the **Password Encoder** dialog.
4. Change the appropriate code lines in your test scripts.



Note: You can also encrypt a string in code by using the `encrypt` method. Encrypted strings can be decrypted using the `decrypt` method.

Using Selenium WebDriver

Selenium WebDriver is a powerful open-source automation tool, enabling automated web-application testing on any browser using a WebDriver-compliant driver. By using Silk4J with WebDriver, you gain the following main benefits:

- You can easily record new WebDriver scripts, instead of writing code.
- You can replay your existing WebDriver scripts with Silk4J, or even schedule them for execution with Silk Central.
- You can generate TrueLog files for your WebDriver scripts. With TrueLog, Silk4J enables you to easily locate the line in your script that generated an error.

Using Selenium with Existing Silk4J Scripts



Note: The functionality described in this topic requires Java 8 or later.

To add WebDriver functionality to an existing Silk4J script, you can create a mixed script. To create such a mixed script, use the Open Agent as a Selenium server which is activated when a Silk4J base state is executed.

1. Add the *Silk Test Selenium Client Library* to the project that contains the Silk4J script.
 - a) In the **Package Explorer**, right-click on the project node.
 - b) Select **Silk4J Tools > Add WebDriver Capability**.
2. Open the Silk4J script.
3. Retrieve the WebDriver ID of the browser instance that you want to refer to:

```
BrowserApplication browserApplication = desktop.find("//  
BrowserApplication");  
WebDriver driver = browserApplication.getWebDriver();
```



Note: If you want to test a hybrid application, use the following code:

```
BrowserWindow browserWindow = desktop.find("//BrowserWindow");  
WebDriver driver = browserWindow.getWebDriver();
```

You can now use the `RemoteWebDriver` object in the same way as when you are using standalone Selenium. For example:

```
driver.get("http://demo.borland.com/InsuranceWebExtJS/");  
driver.findElementById("login-form:login");
```

When you execute the script, all Selenium and Silk4J actions, along with any screenshots and parameters, are logged to the TrueLog.

Executing Selenium Scripts

You can execute Selenium WebDriver scripts with Silk4J to use synchronization and to create a detailed TrueLog during test execution. If a browser type is specified in the capabilities on a machine on which the Open Agent is running, and you create a `RemoteWebDriver` and connect it to the Selenium server which is running on the Open Agent, Silk4J automatically launches the corresponding browser. If the browser type is not specified, the Open Agent tries to reuse an existing browser instance that has been launched in advance for example by executing a Silk4J base state.

To generate a visual execution TrueLog file during the execution of a Selenium script, you can use the *Silk Test TrueLog API for Selenium WebDriver*. This API is implemented as a REST interface, and the endpoint

host and port used by this API are identical to the ones used by the Selenium server, for example `http://localhost:4444/silktest/trueLog`.

The REST API files are located in the Silk Test installation folder under `\ng\TrueLogAPI\`:

- The file `SilkTestTrueLogService-doc.html` contains the REST API documentation.
- The file `SilkTestTrueLogService.yaml` contains the REST API declaration.
- The file `trueLogApiClient.jar` contains a Java client for the REST API.

By importing the REST API declaration file into the [Swagger Editor](#) and then using the **Generate Client** functionality of the editor, without making any changes to the declaration file, you can generate the client API library for a vast amount of languages like Python, Ruby, JavaScript, C#, and others. You can then download the TrueLog API in the selected programming language.

You can execute Selenium scripts with Silk4J against the following browsers:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Apple Safari

1. Start the Open Agent on the machine on which Silk4J is installed.
2. Start the browser against which you want to test your application.

- For Google Chrome, Microsoft Edge, and Mozilla Firefox, you can use the default capabilities from WebDriver to start the browser. For example, you can start Google Chrome by using the java bindings as shown in the following code:

```
RemoteWebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), DesiredCapabilities.chrome());
```

- For Apple Safari, you have to specify a custom browser name as a capability, as shown in the following code:

```
DesiredCapabilities safari = new DesiredCapabilities();  
safari.setCapability("browserName", "SilkSafari");
```

3. *Optional*: Pass additional options as capabilities.

The custom Silk4J options are passed as a map and have the capability name `silkTestOptions`.

For example, the following code sample shows how you can activate the automatic synchronization for Selenium, by setting the **syncEnabled** option:

```
Map<String, Object> options = new HashMap<>();  
options.put("syncEnabled", true);  
capabilities.setCapability("silkTestOptions", options);
```

The following options are allowed:

Option	Type	Description
commandLineArguments	String	Passes command line arguments to the browser that is launched.
connectionString	String	Specifies a connection string to a browser running on a remote machine.
startUrl	String	The URL that the browser navigates to when the browser is launched.
syncEnabled	boolean	Turns the Silk Test AJAX synchronization on or off. The default value is false.
trueLogEnabled	boolean	You can use this option to enable or disable TrueLog. The default value is true.

Option	Type	Description
trueLogId	String	The identifier of the TrueLog session that is returned when calling the <code>StartTrueLog</code> method of the TrueLog API. This identifier is required if you want to specify the TrueLog that should include the WebDriver actions for the Open Agent.
trueLogPath	String	The custom Truelog file path. By default, Truelogs are written to the Silk Test log directory under <code>%LOCALAPPDATA%\Silk\SilkTest\logs</code> .
trueLogScreenshotMode	String	Specifies when screenshots are added to Truelogs. OnError A screenshot is added to the Truelog when an error occurs. always A screenshot is added to the Truelog for each action.

4. *Optional:* To specify the port that is used by the Selenium server, perform the following:

- a) Navigate to `%APPDATA%\Silk\SilkTest\conf`.
- b) Create a new properties file with the name `selenium.properties`.
- c) Type `selenium.server.port=<port name>` into the new file.

The following code sample turns on synchronization and starts Google Chrome with the Selenium Java bindings:

```
DesiredCapabilities capabilities = DesiredCapabilities.chrome();
Map<String, Object> options = new HashMap<>();
options.put("syncEnabled", true);
capabilities.setCapability("silkTestOptions", options);
RemoteWebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capabilities);
```

The following code sample specifies the TrueLog file for the Open Agent and starts Google Chrome with the Selenium Java bindings:

```
TrueLogAPI trueLogAPI = new TrueLogAPI();
trueLogAPI.startTrueLog("C:/temp/myTrueLogFile.tlz");
String trueLogId = trueLogAPI.getTrueLogId();

DesiredCapabilities capabilities = DesiredCapabilities.chrome();
Map<String, Object> options = new HashMap<>();
options.put("trueLogId", trueLogId);
capabilities.setCapability("silkTestOptions", options);
RemoteWebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capabilities);
```

You can now use the `RemoteWebDriver` object in the same way as when you are using standalone Selenium. For example:

```
driver.get("http://demo.borland.com/InsuranceWebExtJS/");
driver.findElementById("login-form:login");
```

When you execute the script, all Selenium and Silk4J actions, along with any screenshots and parameters, are logged to the TrueLog.

Entering Special Keys Into A Text Field

When testing web applications, Silk4J offers the following recording modes:

- **Silk Test**, which allows you to record Silk4J locators.
- **WebDriver**, which allows you to record WebDriver locators. This recording mode is not supported when recording against Internet Explorer.

In this topic, we will describe how you can enter special keys by adding code to a script that has been recorded by using the **WebDriver** recording mode. For information on handling special keys when using the **Silk Test** recording mode, you can refer to the API documentation of the `typeKeys` method.

When using the **WebDriver** recording mode, special keys need to be placed between angled brackets. For example `<back_space>` or `<enter>`.

All special keys in the class [org.openqa.selenium.Keys.java](#) are allowed in Silk4J. The values of the keys are case-insensitive.



Note: In all supported browsers except Mozilla Firefox, you can send multiple special keys or key chords in a single call to the `sendKeys` method. However, to create cross-browser scripts that will also work on Mozilla Firefox, Micro Focus recommends sending only a single string, special key, or key chord at a time with each call to `sendKeys`.

Example

When using the `sendKeys` method on a text field during recording, the following sample parameter values for the `Keys` parameter work on all browsers:

K	K	Generated Java Code
e	ey	
y	s	
s	P	
P	ar	
a	a	
r	m	
a	et	
m	er	
e	Ty	
t	pe	
e		
r		
h	Si	<code>driver.findElement(By.id("login-form:email")).sendKeys("hello");</code>
e	m	
l	pl	
l	e	
o	str	
	in	
	g	
<	S	<code>driver.findElement(By.id("login-form:email")).sendKeys(Keys.BACK_SPACE)</code>
b	pe	<code>;</code>
a	ci	
c	al	
k	ch	
_	ar	
s	ac	
p	ter	
a		
c		
e		
>		
<	C	<code>driver.findElement(By.id("login-form:email")).sendKeys(Keys.chord(Keys.CONTROL, "a"));</code>
c	ho	
o	rd	
n		

Key	Key Parameter Name	Generated Java Code
t		
r		
o		
l		
+		
a		
>		

For example, let us assume we have recorded the following actions.

Recorded Actions
Send keys 'hello'
Send keys '<back_space>'
Send keys ' hello'
Send keys '<control+a>'
Send keys '<back_space>'
Send keys 'bye'

As there are only single special characters or key chords in each call to the `sendKeys` method, these actions replay on all supported browsers, including Mozilla Firefox.

The corresponding generated code, for example in Java, looks like the following:

```
driver.findElement(By.id("login-form:email")).sendKeys("hello");
driver.findElement(By.id("login-form:email")).sendKeys(Keys.BACK_SPACE);
driver.findElement(By.id("login-form:email")).sendKeys("hello");
driver.findElement(By.id("login-form:email")).sendKeys(Keys.chord(Keys.CONTROL, "a"));
driver.findElement(By.id("login-form:email")).sendKeys(Keys.BACK_SPACE);
driver.findElement(By.id("login-form:email")).sendKeys("bye");
```

Using Keyword-Driven Tests as Performance Tests

In addition to performing functional testing and regression testing with Silk4J, you can export your keyword-driven tests to Silk Performer and use them for performance testing and load testing.

To export keyword-driven tests to Silk Performer, ensure that Silk Performer 18.0 or later is installed. Select the Silk4J project that contains the keyword-driven tests and click **Export as Performance Tests** in the Silk4J menu.

If you have already exported the keyword-driven tests in a Silk4J project to Silk Performer, and you want to update the corresponding project in Silk Performer, perform the following actions.

1. Start Silk Performer.
2. Open the project that you want to update.
3. Open the Silk4J project.
4. Click **Export as Performance Tests**.
5. Confirm that you want to update the current Silk Performer project.

The reference to the library in the **Data Files** node of the Silk Performer project will be updated.

For additional information, refer to the Silk Performer documentation.

Known Issues

This section identifies any known issues with Silk4J and their resolutions.

General Issues

Object Map Takes a Long Time to Open

If you have a large object map asset it takes a long time to load when you are using .NET 4. Install .NET 4.5 to resolve this issue.

When a remote desktop or remote desktop connection (RDC) is minimized, Silk Test does not function

When you connect through the remote desktop protocol (RDP) to a desktop, you take ownership of the desktop by attaching to the desktop with your mouse and keyboard. If the desktop is minimized without ownership of the desktop being released, any playback of mouse clicks or keystrokes is undefined.

As a workaround, you could use a VNC-based remote viewing tool. This would allow replay to continue even if the client window is minimized.

The Open Agent does not start when the Check Point firewall is installed

When you have a Check Point firewall or a Check Point ZoneAlarm firewall installed on your system, the Open Agent cannot be started, because the firewall interrupts the communication between the Agent and the infoservice.

To start the Open Agent, you have to uninstall the Check Point firewall from your system.

The `modifiers` parameter in the `domDoubleClick` method is ignored

You cannot specify the modifier in the overloaded `domDoubleClick` method. The modifier will not be double-clicked, although you have specified the parameter. The overloaded `domDoubleClick` method, which allows you to specify the modifier, is deprecated. To specify the modifier, you can use the `doubleClick` method, if you are using a client that supports an overloaded method with the `modifiers` parameter, or the `PressKeys` and `ReleaseKeys` methods.

Silk Test does not support testing Metro-style apps

Silk Test does not support testing Metro-style apps on Microsoft Windows 8, Microsoft Windows 8.1, or Microsoft Windows 10. Metro-style apps are also known as Windows 8 style, Modern UI style, Windows Store style, or Universal Windows Platform (UWP) apps.

The built-in spell checking in Microsoft Windows 8 might interfere with the replay of tests

The built-in spell checking in Microsoft Windows 8 can be enabled in applications like Internet Explorer 10.

If a word was incorrectly spelled during recording, and you replay typing this word, the spell checker will either mark it, or for commonly misspelled words will automatically fix it, which is the same behaviour a real user would get. If your tests were created on an operating system that did not include the spell checking feature, you might get unexpected results when replaying the tests on Microsoft Windows 8. To disable the spell checking, you can do the following:

1. Press **Windows Key + C**.

2. On the Charm bar, click **Settings**.
3. Select **More PC Settings**.
4. Select **General** to see the Spelling selections.



Note: These are system-wide settings, not settings specific to Internet Explorer.

5. Set **Autocorrect misspelled words** to off.
6. Set **Highlight misspelled words** to off.

When a .NET application is started from DevPartner Studio (DPS), Silk Test might not recognize it

To resolve the issue, perform the following steps:

1. Go to the Silk Test installation folder (by default, it's located at: C:\Program Files\SilkTest\SilkTest).
2. For Windows Forms applications, go to ng\agent\plugins\com.borland.fastxd.techdomain.windowsforms.agent_<version number>.
3. For Windows Presentation Foundation (WPF) applications, go to ng\agent\plugins\com.microfocus.silktest.techdomain.wpf.agent_<version number>.
4. In Notepad, open the file plugin.xml, and add the following line to the <loadparameters> section:

```
<param name="frameworkAssembly">mscorlib.dll</param>
```
5. Log out of the computer, and then log back in. Silk Test works as expected with the application that was started by DevPartner Studio.

The highlighting rectangle is out of place when recording clicks on an area of an image

When you record a click on a part of a complex image, for example an area map, the green highlighting rectangle does not highlight the appropriate area of the image. However, the click will be executed correctly during replay.

The Open Agent might not start if Windows Defender is enabled during the installation of Silk Test

If Windows Defender is enabled on your system during the Silk Test installation, you might not be able to start the Open Agent after the installation is complete. Windows Defender might prevent the hotfix setup from performing some required actions. As a workaround, disable Windows Defender during the Silk Test installation.

The IME editor opens out of place when opened from specific locations

The IME editor opens on the top-left corner of the current screen instead of opening near the text field from which it is opened.

This behavior occurs when opening the IME editor from the following locations:

- The **Silk Recorder**.
- The **Keyword-Driven Test Editor**.
- The **Keywords** view.

Cannot use Shift+Insert when the Numeric Lock key is active

Using the `typeKeys` method to paste the contents of the clipboard with **Left Shift+Insert** or **Right Shift+Insert** does not work when the Numeric Lock (Num Lock) key is active.

You can workaround this issue by deactivating the Num Lock in your test script before calling the `typeKeys` method.

Why do I get the error Windows DLL failed to load during installation?

If you are using Java 8 update 212 or a later update of Java 8, installing Silk Test on a Windows Server 2019 machine will fail with the following error: Windows DLL failed to load.

You can work around this issue by downgrading to a Java 8 version prior to update 212 and by starting the installer with a command line call similar to the following:

```
SilkTest<Version>.exe LAX_VM "[path to java.exe]"
```

For example:

```
SilkTest200_64bit.exe LAX_VM "c:\java\bin\java.exe"
```

For additional information on installing Silk Test through the command line, refer to the [Silk Test Installation Guide](#).

Mobile Web Applications

Silk4J does not support frames on Apple Safari

Silk4J does not support HTML frames and iframes on Apple Safari on iOS.

Chrome for Android 43 or later: Silk4J does not support zooming and scrolling at the same time

Recording a mobile web application on Chrome for Android 43 or later, while the mobile web application is zoomed and the top-left corner is not visible on the screen, might not work as expected. If the green rectangles for the controls in a mobile web application are not correctly displayed during recording, zoom-out the mobile web application completely, scroll to the top-left corner of the mobile web application, and refresh the **Recording** window.

Web Applications

Recording with a zoom level different to 100% might not work properly

Recording a Web application with a zoom level different to 100% might not work as expected. Before recording actions against a Web application, set the zoom level in the browser to 100%.

Google Chrome

Locator recording in windows fails with Google Chrome

When you are testing a Web application in Google Chrome, locator recording in windows fails when multiple windows are open during application configuration in the Google Chrome instance, in which the application is running. If you close the other Google Chrome windows during application configuration, the error will not appear.

Background applications in Google Chrome prevent automation support from loading

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.

Silk Test cannot record locators in modal dialogs when Windows Aero is disabled

If Windows Aero functionality is disabled, modal dialogs are not recognized and locators in such dialogs cannot be selected. As a workaround, use the **Locator Spy** or the **Identify Object** dialog box to manually create and verify locators while a modal dialog is displayed.

Silk Test does not display embedded PDFs

With Google Chrome 42 or later, Google Chrome by default blocks the NPAPI plug-in, which is used to display embedded PDFs. Because of this, Silk Test does not display embedded PDFs in Google Chrome 42 or later, but instead downloads the embedded PDFs.

- If you are using Google Chrome 44 or prior, you can unblock the NPAPI plug-in in Google Chrome, by typing the following into the address bar:

```
chrome://flags/#enable-npapi
```

- If you are using Google Chrome 45 or later, the NPAPI plug-in is completely removed from Google Chrome, without an option to re-enable it, and all PDFs are downloaded.

Connection timeouts when executing tests against Google Chrome 49 or prior

When executing tests against Google Chrome 49 or prior on a slow machine, you might experience connection timeouts, causing the tests to fail. The following error message is displayed:

Error executing '*'. Communication with browser automation timed out.

To avoid such connection timeouts, ensure that the test machine has enough processing power. For example, if you are testing on a slow virtual machine (VM), you could enhance the processing power by adding an additional CPU core to the VM.

Setting the UserDataDir through the registry breaks the Google Chrome support when using Google Chrome 66 or later

If the user data directory is set as a policy in the registry through the key `HKEY_LOCAL_MACHINE\Software\Policies\Google\Chrome\UserDataDir` or the key `HKEY_CURRENT_USER\Software\Policies\Google\Chrome\UserDataDir`, and you are testing a web application on Google Chrome 66 or later, the base state might fail with the following error message: Failed to start application 'GoogleChrome'. unknown error: DevToolsActivePort file doesn't exist . This is a known issue in ChromeDriver: <https://bugs.chromium.org/p/chromedriver/issues/detail?id=2513>.

As a workaround, perform one of the following:

- Remove the registry key.



Note: This is the only way to enable parallel testing with Google Chrome if this issue occurs.

- Set the user data directory in Google Chrome to the same directory as in the registry key.

1. Open the **Edit Browser Application Configuration** dialog.

2. Select **Google Chrome** as the browser type.

3. Set the user data directory in the **Connection String** field: `goog:chromeOptions={"args":["--user-data-dir=<user data directory>"]}`. For example, if the value in the registry is `C:/temp/chromeUserData`, type `goog:chromeOptions={"args":["--user-data-dir=C:/temp/chromeUserData"]}`.



Note: Parallel testing with Google Chrome will not work, even if you apply this workaround.

Internet Explorer

Using Google toolbar interferes with recording Web applications

Using the Google toolbar with Internet Explorer 8 interferes with recording locators for Web applications. Turn off the Google toolbar before you record Web applications.

Microsoft Silverlight Applications

Some Microsoft Silverlight applications cause Internet Explorer to hang when interacting with Silk Test. On 32-bit platforms, refer to MS KB 2564958 (an update to Active Accessibility) to help prevent the issue.

Locators recorded with Silk Test versions prior to Silk Test 13.5 might no longer work in Internet Explorer

In Silk Test 13.5, we have adapted the normalization of white spaces of the `textContent` attribute in Internet Explorer. This change was made to improve the cross-browser capabilities of Silk Test, and might affect locators which rely on the `textContent` attribute, and which are used in scripts that were recorded with releases prior to Silk Test 13.5.

The Open Agent cannot have high elevation enabled when UAC is enabled on Microsoft Windows 8 or later and Internet Explorer 11

You cannot test a Web application in Internet Explorer 11 on Microsoft Windows 8 or later and have UAC enabled and run both Internet Explorer and the Open Agent with high elevation.

Known issues with Input Method Editors (IMEs)

- Silk Test does not record half-width spaces for Japanese input in Internet Explorer 11.
- Silk Test does not record IME input in Internet Explorer 11 in compatibility mode.
- In Japanese IME mode, pressing **Space** will cause Silk Test to record the current IME candidate. Use **Convert** to avoid this issue.

Internet Explorer might stop working while testing an Applet with a Java version higher than 1.7 update 71

Internet Explorer might stop working while testing an Applet with a Java version higher than 1.7 update 71 (7u71).

Internet Explorer does not respond during test execution

Internet Explorer 10 or later might become unresponsive during test execution because a thread in Internet Explorer is suspended and creates a deadlock. The root cause of this problem is a new security feature in Internet Explorer.

To solve this issue, disable the security feature.

1. Open the **Registry Editor**.
2. If the **OverrideMemoryProtectionSetting** entry does not exist in the key **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Main**, create the entry.
3. Set the value of the registry key **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Main::OverrideMemoryProtectionSetting** to 2.

The JavaScript alert-handling API methods do not work with an embedded Internet Explorer

The following JavaScript alert-handling methods of the `BrowserWindow` class do not work when testing an embedded Internet Explorer:

- `acceptAlert` method
- `dismissAlert` method
- `getAlertText` method
- `isAlertPresent` method

Microsoft Edge

Windows opened out of Microsoft Edge are not supported

Windows which are opened as actual new Microsoft Edge windows, and not as tabs, are not supported. Silk4J cannot close such windows correctly, and the agent is in an invalid state after closing such a window.

JavaScript alerts opened in an iframe or frame cannot be closed

Microsoft Edge cannot close the JavaScript alerts `alert()`, `prompt()`, and `confirm()`, if these alerts were opened in an iframe or frame.

Cannot open the native browser context menu

If a test includes a right click in Microsoft Edge, which opens the native browser context menu, the test will hang. This issue does not occur with HTML menus opened by Microsoft Edge.

Cannot execute tests against Microsoft Edge with UAC disabled

When executing tests against Microsoft Edge with UAC disabled, the following error message displays: Failed to start application 'Edge'. Unable to parse remote response: Unknown error

To solve this issue, enable UAC.

Why do I get the error "Cannot start Edge because the Windows Feature Microsoft WebDriver is not installed"?

If your system uses the Windows 10 October 2018 Update or a later version of Microsoft Windows 10, and the Windows Feature Microsoft WebDriver is not installed on your system, you might get the following error when trying to test on Microsoft Edge:

Cannot start Edge because the Windows Feature Microsoft WebDriver is not installed. See the topic Known Issues > Microsoft Edge in the Help.

To solve this issue, ensure that the Windows update on your system is working properly. If you are using an internal Windows Update server, perform the following actions:

1. Ask your IT administrator to add the Windows Feature Microsoft WebDriver to the Windows update server.
2. Install the Windows Feature Microsoft WebDriver. For additional information on installing the feature, refer to [Microsoft WebDriver](#).

Mozilla Firefox

Calls into applications using Adobe FlashPlayer do not properly synchronize when using Mozilla Firefox

When you are using Mozilla Firefox with a recent Adobe FlashPlayer version, some calls might not synchronize properly. The following issues might occur:

- Mozilla Firefox might falsely recognize a running script as stalled and a confirmation dialog box might appear asking whether you want to continue the execution of the script, even though the script is running properly.

- Typing characters might not work because `SetFocus` is no longer working correctly.
- The Adobe automation might return an old value although the UI already shows a new value.

If you face one or more of these issues with applications using Adobe FlashPlayer, turn off the ProtectedMode in Adobe FlashPlayer. For additional information, see <http://forums.adobe.com/thread/1018071> and read the information provided under *Last Resort*.

SAP Applications

HierarchyHeaderWidth and ColumnOrder Properties of the SAPTree Class are write only

Other than the automation documentation indicates, the `HierarchyHeaderWidth` and `ColumnOrder` properties of the `SAPTree` class are write only and cannot be read.

Ensure that your scripts use write rather than read with these properties.

GetColumnIndexFromName() of the SapTree Class May Fail with an "unspecified error"

`GetColumnIndexFromName()` of the `SapTree` class may fail with an "unspecified error". This is a known issue in SAP automation.

Check the SAP web site to see if the issue has been resolved.

Calling the select() method of the SAPTree Class on a Context menu item may fail

Calling the `Select()` method of the `SAPTree` class on a Context menu item may fail.

Call `SelectContextMenuItem` on the parent control instead. This problem is a known issue in the SAP automation.

The position property for a horizontal scrollbar always returns 1

The position property for a horizontal scrollbar always returns 1. This is a known issue in SAP automation.

Check the SAP web site to see if the issue has been resolved.

The SAPNetPlan class is not supported

This issue will be resolved in a future release.

Replay error occurs when executing an SAP script

In certain cases, if you record an SAP test and then replay it, the following error might occur: The data necessary to complete this operation is not yet available. This means that Silk4J is executing the recorded actions too fast.

To solve this issue, you can add sleeps to your test script or you can increase the post replay delay to increase the time that Silk4J waits after invoking a function or setting a property. For additional information about this option, see *Agent Options*. You could also change the script to use SAP automation to replay the problematic action instead of using the xBrowser technology domain. For example, you could change the action `DomLink.Select` to `SapHTMLViewer.SapEvent`.

The method getCurrentRow returns a wrong value with SAPGUI client 7.30

If you use SAPGUI client 7.30 and you call the method `getCurrentRow`, the method might falsely return -1 instead of the row number.

The method `resizeWorkingPane` is not working correctly with SAPGUI client 7.30

If you use SAPGUI client 7.30 and you call the method `resizeWorkingPaneEx`, the method will not resize the `workingPane` and calling `getSapWindow().getWidth()` will return a wrong value for the window width.

Silk4J

When you are using object maps, existing locators that do not start with a slash will no longer work

Locators that include only a class name and that do not start with a slash, for example `PushButton`, will no longer work if object maps exist. This issue might result in breaking existing scripts that were created in a Silk Test version prior to Silk Test 14.0. For the previous example the script will fail with the following error:

Identifier 'PushButton' was not found in the Object Map.

More complex locators that include more than a class name, for example `PushButton[@caption=OK]` will continue to work, even if object maps exist.

To fix this issue, add a `//` to the start of any such locator. For example, if the locator `PushButton` in the following code does no longer work:

```
PushButton button = mainWindow.find("PushButton");
```

Change the code to:

```
PushButton button = mainWindow.find("//PushButton");
```

Enabling or Disabling Usage Data Collection

To help Micro Focus improve your overall testing experience, Micro Focus would like to collect some information on how you use Micro Focus software and services. By accepting the terms of the License Agreement while installing Silk4J, you allow Micro Focus to collect information about how you use Silk4J and about the computer system on which Silk4J is installed. Micro Focus does not collect any personally identifiable information, for example your name or address, or any of your data files, for example scripts or passwords. By allowing Micro Focus to collect this information, you assist Micro Focus in identifying trends and usage patterns.

To enable or disable the collection of usage data by Micro Focus:

1. Click **Silk4J > About Silk4J** in the menu.
2. In the About dialog box, click **Customer Feedback Options**.
3. Select one of the following options:
 - To enable usage data collection, click **Yes, I am willing to participate**.
 - To disable usage data collection, click **No, I would not like to participate**.
4. Click **OK**.

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
 - overview 163
 - Silverlight 175
 - Windows Forms overview 163
 - Windows Presentation Foundation (WPF) overview 167

A

- Accessibility
 - enabling 88, 295
 - improving object recognition 295
 - using 295
- action recording
 - merging object map entries 266
- ActiveX
 - invoking methods 91
 - overview 91
- add-ons
 - Google Chrome 203
 - Mozilla Firefox 206
- adding
 - keywords 235
- adding keywords
 - keyword-driven tests 234
- adding multiple images
 - image assets 278
- admin rights
 - installing 12
 - running 12
- Adobe Flex
 - adding configuration information 108
 - Adobe Air support 104
 - automationName property 111
 - coding containers 114
 - containers 114
 - creating applications 109
 - FlexDataGrid control 105
 - invoking methods 94
 - loading at run-time 108
 - multiview containers 115
 - passing parameters 108
 - passing parameters at runtime 108
 - passing parameters before runtime 108
 - run-time loading 107
 - security settings 117
 - select method, overview 104
 - select method, setting 113
 - testing playback 116
- advanced
 - options 88
- agent options
 - Open Agent 54
- agents
 - configuring ports, information service 64
 - options 54
 - overview 54
 - port numbers 62
 - starting 54
- AJAX applications
 - browser recording options, setting 82
 - script hangs 216
- analyzing
 - test results 51
- analyzing results
 - tests 51
- Android
 - configuring emulator 127
 - enabling USB-debugging 127
 - hybrid applications 125
 - installing USB drivers 126
 - invoking methods 161
 - mobile native applications, prerequisites 124
 - mobile web applications, prerequisites 124
 - native apps, creating tests 26
 - parallel testing, tested configurations 129
 - recommended settings 127
 - releasing devices 151
 - releasing devices, recording 151
 - releasing devices, replay 151
 - testing 124
 - troubleshooting 152
 - web applications, creating tests 25
- Android emulator
 - configuring 127
- Ant
 - replaying tests, troubleshooting 38
 - running keyword-driven tests 242
 - running tests 36
- Apache Flex
 - Component Explorer 93
 - attributes 117, 220
 - automationIndex property 110
 - automationName property 110
 - class definition file 102, 111
 - controls are not recognized 117
 - custom controls 93, 291
 - custom controls, defining 95, 102, 111
 - custom controls, implementing 100
 - customizing scripts 103
 - enabling your application 105
 - Flash player settings 92
 - initializing, applications 115
 - invoking methods 94
 - invoking methods for custom controls 97
 - linking automation packages 105
 - overview 92
 - precompiling the application 106
 - recording, applications 115
 - select method, setting 113
 - styles 116
 - testing 93
 - testing multiple applications 103
 - workflow 115
- Apache Flex applications

- custom attributes 110, 259
- API playback
 - comparing to, native playback 191
- Apple Safari
 - connection string 194
 - information service, installing 137, 142, 198, 201
 - limitations 199
 - preparing 198
 - prerequisites 197
 - running multiple tests 201
 - support 186
 - testing 197
- application configurations
 - definition 74
 - deleting 75
 - errors 78
 - keyword-driven tests 232
 - modifying 75
 - removing 75
 - troubleshooting 78
- assets
 - opening from a script 279
- attribute types
 - Apache Flex 117, 220
 - Java AWT 118, 221
 - Java Swing 118, 221
 - Java SWT 122, 221
 - Oracle Forms 121
 - overview 220
 - SAP 181, 221
 - Silverlight 175, 222
 - UI Automation 223, 300
 - Web applications 219, 224
 - Windows 168, 225
 - Windows Forms 163, 224
 - xBrowser 219, 224
- attribute values
 - finding with Locator Spy 32

B

- base state
 - definition 69
 - executing 72
 - keyword-driven tests 232
 - modifying, script 71
 - modifying, user interface 69
- basestate
 - about 69
- best practices
 - scripts, creating 30
- blacklists
 - UI Automation, locator attributes 87
- browser
 - defining, playback 187
 - maximize 214
- browser configuration settings
 - xBrowser 192
- browser testing
 - replay, parallel 46
- browser type
 - Chrome for Android, setting 157

- viewing current 215
 - viewing current, GetProperty 215
- browser window
 - specifying size 209
- browsers
 - options, setting 82
 - starting, scripts 218
- browsertype
 - Chrome for Android, setting 157
 - using 215

C

- calling dlls
 - example 283
 - Java 282
 - scripts 282
- capabilities
 - iOS 66, 141
- capturing
 - web pages, full page 53
- CEF
 - testing 185
- certificates
 - installing, Mobile Center 146
 - replacing, information service 67
- Chrome
 - configuration settings 192
 - cross-browser scripts 217
 - extensions, testing 203
 - known issues 322
 - locators 217
 - prerequisites 202
 - recording tests 29
 - testing 201
 - user data directories, testing 203
- Chrome for Android
 - browser type, setting 157
 - support 186
- Chromium Embedded Framework
 - testing 185
- class names
 - finding with Locator Spy 32
- classes
 - exposing 85
 - ignoring 85
- Click
 - mobile web 162
- combining
 - keywords 235
- command line
 - running keyword-driven tests 241
 - running tests 35
- Component Explorer
 - Apache Flex 93
- Configuration Assistant
 - automatic signing 138
- configuration testing
 - Silk Central Connect 310
- configuring ports
 - information service, clients 64
 - Open Agent 65

- connection string
 - desktop browsers, local 196
 - desktop browsers, remote 194
 - mobile devices 147
 - contact information 329
 - continuous integration
 - uploading keyword libraries 248
 - continuous integration servers
 - running tests 38
 - running tests on Silk Central 45
 - creating
 - keyword-driven tests 230
 - creating stable locators
 - overview 257
 - creating tests
 - mobile native applications 26
 - mobile web applications 25
 - standard applications 25
 - web applications 24
 - creating visual execution logs
 - TrueLog 51
 - TrueLog Explorer 51
 - cross browser testing
 - Apple Safari 197
 - Apple Safari, limitations 199
 - current browser type, viewing 215
 - FAQs 214
 - Google Chrome 201
 - Microsoft Edge 208
 - Microsoft Edge, limitations 208
 - mouse move preferences, setting 192
 - Mozilla Firefox 205
 - object maps, using 267
 - object recognition 189
 - overview 186
 - recording locators 217
 - scrolling 214
 - test objects 189
 - wrong timestamps, logs 215
 - cross-browser testing
 - Apple Safari 197
 - Apple Safari, limitations 199
 - connection string 194
 - current browser type, viewing 215
 - FAQs 214
 - Google Chrome 201
 - Microsoft Edge 208
 - Microsoft Edge, limitations 208
 - mouse move preferences, setting 192
 - Mozilla Firefox 205
 - object maps, using 267
 - object recognition 189
 - overview 186
 - recording locators 217
 - remote locations, adding 77
 - scrolling 214
 - test objects 189
 - wrong timestamps, logs 215
 - current browser type
 - viewing 215
 - custom attributes
 - Apache Flex applications 110, 259
 - controls 259
 - including in tests 33
 - setting 84
 - Web applications 219, 260
 - Windows Forms applications 164, 261
 - WPF applications 169, 261
 - custom controls
 - adding code to AUT 288
 - adding code to AUT, FAQs 290
 - Apache Flex, defining 102, 111
 - Apache Flex, implementing 100
 - creating custom classes 293
 - dialog box 294
 - dynamic invoke, FAQs 288
 - dynamically invoking Apache Flex 97
 - Flex, defining 95
 - injected code is not used in AUT 290
 - invoke call returns unexpected string 288
 - managing 291
 - overview 287
 - supporting 293
 - testing (Apache Flex) 93, 291
 - custom properties
 - controls 259
 - Customer Care 329
- ## D
- debug
 - Docker 44
 - deleting
 - application configurations 75
 - keywords 235
 - deleting scripts
 - object map items 275
 - device not connected
 - mobile 152
 - Dialog
 - not recognized 214
 - discovering endpoints
 - service virtualization 304
 - dlls
 - aliasing names 286
 - calling conventions 286
 - calling from Java 282
 - calling from within a script 282
 - example call 283
 - function declaration syntax 283
 - passing arguments that can be modified to functions 285
 - passing arguments to functions 284
 - passing string arguments to functions 285
 - Docker
 - environment variables 40
 - example 41
 - limitations 43
 - tests, running 38
 - troubleshooting 44
 - docker-compose
 - example 42
 - downloads 329
 - dynamic invoke

- adding code to AUT, FAQs 290
- Android 161
- FAQs 288
- input argument types do not match 290
- iOS 161
- mobile native 161
- overview 287
- simplify scripts 288
- unexpected return value 288
- dynamic locator attributes
 - about 226
- dynamic object recognition
 - creating test 30
- dynamically invoking methods
 - ActiveX 91
 - Apache Flex 94
 - Apache Flex custom controls 97
 - Java AWT 118, 122
 - Java Swing 118, 122
 - Java SWT 118, 122
 - SAP 181
 - SAP controls 182
 - Silverlight 176
 - UI Automation 298
 - Visual Basic 91
 - Windows Forms 164
 - Windows Presentation Foundation (WPF) 170
- DynamicInvoke
 - ActiveX 91
 - Android 161
 - Apache Flex 94
 - iOS 161
 - Java AWT 118, 122
 - Java Swing 118, 122
 - SAP 181
 - Silverlight 176
 - UI Automation 298
 - Visual Basic 91
 - Windows Forms 164
 - Windows Presentation Foundation (WPF) 170

E

- Eclipse
 - troubleshooting 123
- Eclipse RCP
 - support 121
- Edge
 - connection string 194
 - known issues 325
 - limitations 208
 - locators 217
 - recording tests 27
 - remote testing 195
 - testing 208
- editing
 - application configurations 75
 - remote locations 77
- editing password
 - Mobile Center 147
- embedded Chrome
 - testing 185
- Emulator
 - defining, playback 143

- emulators
 - testing 124
- enabling TrueLog
 - TrueLog Explorer 52
- encode
 - passwords 312
- encrypt
 - passwords 312
- environment variables
 - Docker 40
- exclude lists
 - UI Automation 87
- excluded characters
 - recording 34
 - replay 34
- executing keyword-driven tests
 - variables 244
- exposing
 - WPF classes 85
- extensions
 - Google Chrome 203
 - Mozilla Firefox 206

F

- FAQs
 - adding code, AUT 290
 - cross-browser testing 214
 - dynamic invoke 288
 - object maps 274
- filtering
 - keywords 250
- find references
 - keywords 250
- Firefox
 - configuration settings 192
 - cross-browser scripts 217
 - extensions, testing 206
 - limitations 206, 207
 - locators 217
 - profiles, testing 205
 - recording tests 28
 - testing 205
- firewalls
 - port numbers 64
 - resolving conflicts 62
- Flash player
 - opening applications in 92
 - security settings 117
- Flex
 - adding configuration information 108
 - Adobe Air support 104
 - attributes 117, 220
 - automationIndex property 110
 - automationName property 110, 111
 - class definition file 102, 111
 - Component Explorer 93
 - containers 114
 - creating applications 109
 - custom controls 93, 291
 - custom controls, defining 95, 102, 111
 - custom controls, implementing 100

- customizing scripts 103
- enabling your application 105
- Flash player settings 92
- FlexDataGrid control 105
- initializing, applications 115
- invoking methods 94
- invoking methods for custom controls 97
- linking automation packages 105
- loading at run-time 108
- multiview containers 115
- overview 92
- passing parameters 108
- passing parameters at runtime 108
- passing parameters before runtime 108
- precompiling the application 106
- recording, applications 115
- run-time loading 107
- security settings 117
- select method, overview 104
- select method, setting 113
- styles 116
- testing 93
- testing multiple applications 103
- testing playback 116
- workflow 115

- Flex applications
 - creating tests 24
- frequently asked questions
 - adding code, AUT 290
 - cross-browser testing 214
 - dynamic invoke 288
 - object maps 274

- full screen
 - browser 214

G

- Google Chrome
 - additional versions, testing 213
 - capabilities, setting 196
 - configuration settings 192
 - extensions, testing 203
 - full screen 214
 - iframe performance, improving 211
 - known issues 322
 - limitations 203
 - limitations, macOS 204
 - macOS 195
 - options, setting 196
 - prerequisites 202
 - remote testing 195
 - support 186
 - testing 201
 - user data directories, testing 203

- grouping
 - keywords 251
 - object map items 274

- GWT
 - locating controls 258

H

- hidden
 - input fields 219

- HTML reports
 - about 51
 - enabling 81
- HTTPS
 - certificates, Mobile Center 146
 - certificates, replacing 67
 - information service 66, 141
- hybrid applications
 - Android 125
 - iOS 135

I

- identifiers
 - stable 257
- identifying controls
 - dynamic locator attributes 226
 - Locator Spy 262
- identifying objects
 - overview 252
- iframes
 - performance, improving 211
- ignoring
 - classes 85
- image assets
 - creating 277
 - multiple images, adding 278
 - overview 277
 - using in other projects 265, 280
- image checks
 - overview 279
- image click recording
 - overview 276
- image clicks
 - recording 276
- image recognition
 - enabling 276
 - methods 276
 - overview 276
- image verifications
 - adding during recording 280
 - creating 279
 - overview 279
 - using in other projects 265, 280
- IMEs
 - UI Automation 301
- implementing
 - keywords 235
- importing
 - projects 23
- improving object recognition
 - Accessibility 295
- information service
 - certificates, replacing 67
 - configuring ports, clients 64
 - editing 66, 141
 - HTTPS 66, 141
 - Mac, installing 137, 142, 198, 201
 - ports, configure 62
- initializing
 - Apache Flex applications 115
- innerHTML

- xBrowser 216
 - innerText
 - xBrowserf 216
 - input argument types do not match
 - dynamic invoke 290
 - input fields
 - finding 219
 - installing
 - information service, Mac 137, 142, 198, 201
 - privileges required 12
 - installing USB drivers
 - Android 126
 - integrations
 - configuring Silk Central location 245
 - Internet Explorer
 - configuration settings 192
 - cross-browser scripts 217
 - full screen 214
 - known issues 324
 - link.select focus issue 215
 - locators 217
 - misplaced rectangles 216
 - support 186
 - Internet Explorer 10
 - unexpected Click behavior 217
 - invalidated-handle error
 - troubleshooting 218
 - invoke
 - ActiveX 91
 - Android 161
 - iOS 161
 - Java AWT 118, 122
 - Java SWT 118, 122
 - SAP 181
 - Silverlight 176
 - Swing 118, 122
 - UI Automation 298
 - Visual Basic 91
 - Windows Forms 164
 - Windows Presentation Foundation (WPF) 170
 - Invoke method
 - callable methods 288
 - InvokeMethods
 - ActiveX 91
 - Android 161
 - Apache Flex 94
 - iOS 161
 - Java AWT 118, 122
 - Java Swing 118, 122
 - SAP 181
 - Silverlight 176
 - UI Automation 298
 - Visual Basic 91
 - Windows Forms 164
 - Windows Presentation Foundation (WPF) 170
 - iOS
 - apps, preparing for testing 136
 - devices, preparing 136
 - hybrid applications 135
 - information service, installing 137, 142, 198, 201
 - invoking methods 161
 - Mac, preparing 138
 - mobile native applications, prerequisites 131
 - mobile web applications, prerequisites 131
 - native app, Simulator 133
 - native app, testing 132
 - native apps, creating tests 26
 - recommended settings 142
 - releasing devices 151
 - releasing devices, recording 151
 - releasing devices, replay 151
 - testing 130
 - testing, no developer account 139
 - web app, Simulator 134
 - web app, testing 134
 - web applications, creating tests 25
 - iOS 9.3
 - existing scripts, executing 142
- ## J
- Java AWT
 - attribute types 118, 221
 - attributes 118, 221
 - custom attributes 33
 - invoking methods 118, 122
 - overview 118
 - Java AWT/Swing
 - priorLabel 120
 - Java FX
 - support 296
 - Java Network Launching Protocol (JNLP)
 - configuring applications 79, 120
 - Java Swing
 - attributes 118, 221
 - invoking methods 118, 122
 - overview 118
 - Java SWT
 - attribute types 122, 221
 - custom attributes 33, 84
 - invoking methods 118, 122
 - support 121
 - troubleshooting 123
 - Java SWT applications
 - creating tests 25
 - JNLP
 - configuring applications 79, 120
 - JUnit test case
 - creating 30
- ## K
- keyword libraries
 - uploading 248
 - keyword sequences
 - creating 240
 - parameters 238
 - keyword-driven
 - testing 228
 - keyword-driven test editor
 - recommended keywords 237
 - keyword-driven testing
 - advantages 228
 - keyword recommendations, algorithm 237

- marking test methods 234
- overview 228
- parameters, example 238
- troubleshooting 251
- keyword-driven tests
 - adding keywords 234
 - application configurations 232
 - base state 232
 - creating 230
 - editing 234
 - executing from Silk Central 44
 - implementing keywords 232
 - implementing Silk Central keywords 246
 - keywords, searching 250
 - recording 230
 - removing keywords 234
 - replaying 240, 241
 - running from command line 241
 - running from Eclipse 35
 - running with Ant 242
 - specifying variables, execution 244
 - stopping 240
 - stopping during execution, Silk Central 241
 - uploading keywords, Silk Central 246
- keywords
 - about 229
 - adding 235
 - combining 235, 240
 - deleting 235
 - filtering 250
 - find references 250
 - finding in project 250
 - grouping 251
 - implementing 232, 235
 - managing 235
 - marking test methods 234
 - nesting 235
 - opening 235
 - parameters 235, 238
 - parameters, example 238
 - recording 233
 - replacing 235
 - sequences 235
 - uploading to Silk Central 246
- known issues
 - about 320
 - general issues 320
 - Google Chrome 322
 - Internet Explorer 324
 - Microsoft Edge 325
 - mobile web applications 322
 - Mozilla Firefox 325
 - SAP 326
 - Silk4J 327
 - web applications 322
- L**
- learning
 - service virtualization 305
- libraries
 - uploading 248
- licensing
 - available license types 11

- limitations
 - Apple Safari 199
 - Docker 43
 - Google Chrome 203
 - Google Chrome, macOS 204
 - Microsoft Edge 208
 - mobile web applications 158
 - Mozilla Firefox 206, 207
 - native mobile applications 159
 - Windows 8 220
 - Windows 8.1 220
- load testing
 - Silk Performer 319
- LoadAssembly
 - assembly cannot be copied 290
- locating controls
 - GWT example 258
 - siblings example 258
- locator attributes
 - dynamic 226
 - excluded characters 34
 - recording options, setting 82
 - Rumba controls 180, 224
 - Silverlight controls 175, 222
 - UI Automation controls 223, 300
 - WPF controls 168, 225
- locator generator
 - configuring for xBrowser 194
- Locator Spy
 - adding locators to test methods 32
 - adding object map items to test methods 32
 - overview 262
- locators
 - basic concepts 252
 - customizing 257
 - incorrect in xBrowser 216
 - mapping 264
 - modifying in object maps 269
 - object types 252
 - search scopes 252
 - supported constructs 253
 - supported subset 255
 - syntax 253
 - unsupported constructs 253
 - using attributes 253
 - xBrowser 217

M

- Mac
 - Apple Safari, prerequisites 197
 - Apple Safari, testing 197
 - information service, installing 137, 142, 198, 201
- managing
 - keywords 235
- manually creating
 - object maps 275
- maximize
 - browser 214
- merging
 - object maps 275
- MFC

- support 186
- Microsoft Accessibility
 - improving object recognition 295
- Microsoft Edge
 - additional versions, testing 213
 - connection string 194
 - iframe performance, improving 211
 - known issues 325
 - limitations 208
 - recording tests 27
 - remote testing 195
 - support 186
 - testing 208
- Microsoft Foundation Class
 - support 186
- Microsoft UI Automation
 - recording tests 297
- missing peripherals
 - test machines 13
- mobile
 - troubleshooting 152
- mobile applications
 - recording 143
 - testing 124
- mobile apps
 - creating tests 26
- mobile browsers
 - limitations 158
- Mobile Center
 - certificates, installing 146
 - enabling 144
 - enabling, Silk Central 145
- mobile device
 - defining, playback 143
- mobile devices
 - interacting with 151
 - performing actions against 151
- mobile native applications
 - creating tests 26
 - limitations 159
- mobile recording
 - about 143
- mobile testing
 - Android 124
 - connection string 147
 - iOS 130
 - native app, iOS Simulator 133
 - overview 124
 - releasing devices 151
 - remote locations, adding 77
 - replay, parallel 46
 - web app, iOS 134
 - web app, iOS Simulator 134
- mobile testing devices
 - native app, iOS 132
- mobile web
 - Click 162
 - iOS 134
 - known issues 322
 - legacy tests 163
- mobile web applications
 - Android, prerequisites 124

- creating tests 25
- iOS, prerequisites 131
- limitations 158

- mouse move actions
 - recording 82
- mouse move preferences
 - setting, cross-browser testing 192

- Mozilla Firefox
 - additional versions, testing 213
 - capabilities, setting 196
 - configuration settings 192
 - extensions, testing 206
 - full screen 214
 - iframe performance, improving 211
 - known issues 325
 - limitations 206, 207
 - macOS 195
 - options, setting 196
 - profiles, testing 205
 - recording tests 28, 29
 - remote testing 195
 - support 186
 - supported versions, new 17
 - testing 205

- MSUIA
 - invoking methods 298
 - object recognition, improving 296
- multiple agents
 - single machine 311
- multiple applications
 - single machine 311
 - testing 79
- multiple images
 - adding, image assets 278

N

- native mobile
 - invoking methods 161
- native mobile applications
 - Android, prerequisites 124
 - iOS, prerequisites 131
 - limitations 159
- native playback
 - comparing to, API playback 191
- native user input
 - advantages 191
- nesting
 - keywords 235
- network address translation (NAT)
 - configuring 68

O

- object map items
 - adding 271
 - copying 271
 - deleting 273
 - finding errors 273
 - grouping 274
 - highlighting 272
 - identifying 269, 272

- locating in test application 272
- modifying locators 269
- renaming 267
- updating from test application 270
- object maps
 - adding items 271
 - advantages 264
 - benefits 264
 - best practices 274
 - copying items 271
 - deleting items 273
 - deleting, scripts 275
 - FAQs 274
 - grouping items 274
 - manually creating 275
 - merging 275
 - merging during action recording 266
 - modifying 268
 - opening from a script 272
 - overview 264
 - recording 265
 - renaming items 267
 - turning off 265
 - turning on 265
 - using in other projects 265, 280
 - web applications 267
 - xBrowser 267
- object recognition
 - creating stable locators 257
 - custom attributes 259
 - Exists method 256
 - FindAll method 256
 - identifying multiple objects 256
 - improving with Accessibility 295
 - improving, UI Automation 296
 - overview 252
 - using attributes 253
 - UWP, troubleshooting 184
- object types
 - locators 252
- objects
 - checking for existence 256
 - locating 252
- Open Agent
 - configure ports, remote agent 62
 - configuring ports, information service 64
 - connection port, configuring 65
 - network address translation (NAT), configuring 68
 - options 54
 - overview 54
 - port numbers 62
 - starting 54
 - starting from script 54
 - stopping from script 54
 - testing, remote 68
- opening
 - keywords 235
- OPT_ALTERNATE_RECORD_BREAK
 - options 82
- OPT_ASSET_NAMESPACE
 - option 86
- OPT_ENABLE_ACCESSIBILITY

- option 88
- OPT_ENABLE_EMBEDDED_CHROME_SUPPORT
 - options 185
- OPT_ENABLE_MOBILE_WEBVIEW_FALLBACK_SUPPORT
 - option 88
- OPT_ENABLE_UI_AUTOMATION_SUPPORT
 - options 87
- OPT_ENSURE_ACTIVE_OBJDEF
 - option 86
- OPT_IMAGE_ASSET_DEFAULT_ACCURACY
 - option 88
- OPT_IMAGE_VERIFICATION_DEFAULT_ACCURACY
 - option 88
- OPT_LOCATOR_ATTRIBUTES_CASE_SENSITIVE
 - option 88
- OPT_RECORD_MOUSEMOVE_DELAY
 - options 82
- OPT_RECORD_MOUSEMOVES
 - options 82
- OPT_RECORD_SCROLLBAR_ABSOLUT
 - options 82
- OPT_REMOVE_FOCUS_ON_CAPTURE_TEXT
 - option 88
- OPT_REPLAY_MODE
 - option 86
- OPT_RESIZE_APPLICATION_BEFORE_RECORDING
 - options 82
- OPT_WAIT_RESOLVE_OBJDEF
 - options, synchronization 85
- OPT_WAIT_RESOLVE_OBJDEF_RETRY
 - options, synchronization 85
- OPT_XBROWSER_RECORD_LOWLEVEL
 - options 82
- OPT_XBROWSER_SYNC_EXCLUDE_URLS
 - options, synchronization 85
- OPT_XBROWSER_SYNC_MODE
 - options, synchronization 85
- OPT_XBROWSER_SYNC_TIMEOUT
 - options, synchronization 85
- options
 - advanced 88
 - OPT_ENABLE_EMBEDDED_CHROME_SUPPORT
 - 185
 - Playback Status dialog box, setting 49
 - synchronization, setting 85
- Oracle Forms
 - about 120
 - attributes 121
 - prerequisites 121
 - supported versions 120
- ordering
 - tests 45

P

- page synchronization
 - xBrowser 190
- parallel replay
 - browsers 46
 - mobile tests 46
- parallel testing
 - tested configurations, Android 129

- parameters
 - handling, keywords 238
 - Silk Central 310
- password
 - Mobile Center, changing 147
- passwords
 - encrypting 312
- pause recording
 - shortcut key combination 82
- performance testing
 - Silk Performer 319
- playback
 - Playback Status dialog box, setting options 49
 - selecting browser 187
 - selecting device 143
- ports
 - configuring, information service 64
 - Open Agent 62
- pre-fill
 - setting during recording and replaying 174
- preferences
 - turning off error messages 89
- prerequisites
 - Android, mobile web applications 124
 - Android, native mobile applications 124
 - Apple Safari 197
 - Google Chrome 202
 - iOS, mobile web applications 131
 - iOS, native mobile applications 131
- priorLabel
 - Java AWT/Swing technology domain 120
 - Win32 technology domain 184
- privileges required
 - installing Silk Test 12
 - running Silk Test 12
- product suite
 - components 14
- Product Support 329
- profiles
 - Mozilla Firefox 205
- project dependencies
 - adding 265, 280
- project properties
 - converting 90
- projects
 - about 22
 - importing 23

Q

- QT
 - support 296
- Quick Start tutorial
 - introduction 19
 - tests, recording 20
 - tests, replaying 21

R

- recognizing objects
 - xBrowser 189
- recommendations
 - algorithm 237
 - recommended keywords
 - keyword-driven test editor 237
 - recording
 - actions into existing tests 282
 - adding image verifications 280
 - Apache Flex applications 115
 - available actions 31
 - keyword-driven tests 230
 - keywords 233
 - mobile applications 143
 - no image displayed 152
 - object maps 265
 - preferences 82
 - releasing devices 151
 - setting pre-fill 174
 - recording actions
 - existing tests 282
 - recording options
 - browser, setting 82
 - recording tests
 - Google Chrome 29
 - Microsoft Edge 27
 - Microsoft UI Automation 297
 - Mozilla Firefox 28
 - releasing devices
 - mobile testing 151
 - recording 151
 - replay 151
 - remote agent
 - about 68
 - remote browser testing
 - connection string 194
 - remote locations
 - adding 77
 - editing 77
 - remote testing
 - Google Chrome 195
 - Microsoft Edge 195
 - Mozilla Firefox 195
 - Open Agent 68
 - removing keywords
 - keyword-driven tests 234
 - replacing
 - keywords 235
 - replay
 - Dialog not recognized 214
 - options 86
 - Playback Status dialog box, setting options 49
 - releasing devices 151
 - selecting browser 187
 - selecting device 143
 - replaying
 - setting pre-fill 174
 - replaying tests
 - remote machines 68
 - report types
 - selecting 81
 - resolving
 - categories 310
 - responsive web design
 - browser window, specifying size 209

- visual breakpoints, detecting 210
- result comments
 - adding to scripts 310
- results
 - analyzing 51
 - HTML reports 51
- Rumba
 - about 179
 - enabling and disabling support 180
 - locator attributes 180, 224
 - Unix display 180
- Rumba locator attributes
 - identifying controls 180, 224
- running existing scripts
 - iOS 9.3 142
- running multiple tests
 - Apple Safari 201
- running tests
 - continuous integration servers 38
 - Docker 38
 - Silk Central 44
 - stopping 35

S

- Safari
 - connection string 194
 - limitations 199
 - preparing 198
 - prerequisites 197
 - running multiple tests 201
 - testing 197
- SAP
 - attribute types 181, 221
 - custom attributes 84
 - invoking methods 181
 - known issues 326
 - overview 181
 - security settings 183
- SAP controls
 - dynamically invoking methods 182
- SauceLabs
 - enabling 147
- screencast
 - not working 152
- scripts
 - adding result comments 310
 - adding verifications while recording 32
 - creating, best practices 30
 - marking tests as keywords 234
 - object mapping 264
 - specifying options 81
- scroll events
 - recording absolute values 82
- scrolling
 - cross-browser testing 214
- search scopes
 - locators 252
- searching
 - keywords, keyword-driven tests 250
- sections
 - TrueLog 52
- secure connections
 - information service 66, 141

- security settings
 - SAP 183
- select application
 - dialog box 76
- Select method
 - Apache Flex, setting 113
- Selenium
 - about 314
 - mixed scripts, executing 314
 - scripts, executing 314
- serial number 329
- service virtualization
 - about 304
 - discovering endpoints 304
 - learning 305
 - simulating 306
 - simulating, API 306
- setLocation method
 - Android 152
- SetText
 - browser recording options, setting 82
- setting
 - mouse move preferences, cross-browser testing 192
- setting browser
 - playback 187
- setting browser options
 - SetText 82
 - TypeKeys 82
- setting mobile device
 - playback 143
- settings
 - Playback Status dialog box 49
- siblings
 - locating 258
- Silk Central
 - configuring location 245
 - Mobile Center, enabling 145
 - parameters 310
 - running tests 44
 - running tests on continuous integration servers 45
 - SauceLabs, enabling 147
 - uploading keywords 246
- Silk Central Connect
 - configuration testing 310
- Silk Central keywords
 - implementing 246
- Silk Performer
 - measure execution time 311
- Silk4J
 - about 12
 - creating project 19, 22
 - known issues 327
 - quick start tutorial 19
- Silk4J tests
 - grouping 308
- Silverlight
 - attribute types 175, 222
 - invoking methods 176
 - locator attributes 175, 222
 - overview 175
 - scrolling 178
 - support 175

- troubleshooting 178
- simulating
 - service virtualization 306
 - service virtualization, API 306
- Simulator
 - defining, playback 143
 - mobile web applications, testing 134
 - native app, testing 133
 - testing 132
- sleep
 - adding to tests 311
- slowing down
 - tests 311
- special keys
 - recording, WebDriver mode 316
- specifying options
 - scripts 81
- specifying size
 - browser window 209
- stable identifiers
 - about 257
- stable locators
 - creating 257
- standard applications
 - creating tests 25
- starting browsers
 - replay 218
- starting Open Agent
 - scripts 54
- stopping
 - keyword-driven tests, Silk Central 241
 - running keyword-driven tests 240
 - tests 35
- stopping Open Agent
 - scripts 54
- styles
 - in Flex applications 116
- SupportLine 329
- Swing
 - attributes 118, 221
 - configuring JNLP applications 79, 120
 - custom attributes 33
 - invoking methods 118, 122
 - overview 118
- synchronization
 - about 48
 - changing settings 48
 - options, setting 85
 - wrong timestamps 215
 - xBrowser 190

T

- test automation
 - obstacles 13
 - synchronization 48
- test case
 - creating 30
- test machines
 - missing peripherals 13
- test methods
 - adding locators 32

- adding object map items 32
- marking as keywords 234
- test results
 - analyzing 51
- test scripts
 - creating, best practices 30
- testing
 - best practices 12
- testing Apple Safari
 - information service, installing 137, 142, 198, 201
- testing custom controls
 - adding code to AUT 288
- tests
 - analyzing results 51
 - creating 24
 - enhancing 282
 - ordering 45
 - recording actions 282
 - recording, Quick Start tutorial 20
 - replaying, Quick Start tutorial 21
 - running from command line 35
 - running from Eclipse 35
 - running with Ant 36
 - slowing down 311
 - stopping during execution 35
- text click recording
 - overview 302
- text recognition
 - overview 302
- textContents
 - xBrowser 216
- timestamps
 - wrong, cross-browser tests 215
- transparent classes
 - setting 85
- troubleshooting
 - application configurations 78
 - category cannot be resolved 310
 - Eclipse 123
 - invalidated-handle error 218
 - Java SWT 123
 - keyword-driven testing 251
 - mobile 152
 - running tests, Ant 38
 - Silverlight 178
 - UI Automation 302
 - XPath 262
- TrueLog
 - change TrueLog location 52
 - configuring 81
 - creating visual execution logs 51
 - enabling 52, 81
 - replacement characters for non-ASCII 53
 - SilkTestCategory class 308
 - wrong non-ASCII characters 53
- TrueLog Explorer
 - configuring 81
 - creating visual execution logs 51
 - enabling 81
 - enabling TrueLog 52
- TrueLogs
 - sections 52

- tutorial
 - quick start 19
- TypeKeys
 - browser recording options, setting 82

U

- UI Automation
 - attribute types 223, 300
 - exclude lists 87
 - invoking methods 298
 - limitations 301
 - locator attributes 223, 300
 - object recognition, improving 296
 - options, setting 87
 - recording tests 297
 - scrolling 300
 - troubleshooting 302
- unexpected Click behavior
 - Internet Explorer 217
- Unicode content
 - support 296
- Universal Windows Platform
 - support 183
 - troubleshooting 184
- Unix display
 - Rumba 180
- upload app
 - Mac 124
- uploading
 - keyword libraries 248
 - libraries 248
- usage data collection
 - disabling 328
 - enabling 328
- user data directories
 - Google Chrome 203
- UWP
 - troubleshooting 184
- UWP apps
 - support 183

V

- variables
 - executing keyword-driven tests 244
- verification logic
 - adding to scripts while recording 32
- verifications
 - adding to scripts 32
- video
 - not displayed 152
- virtual machines
 - network address translation (NAT), configuring 68
- Visual Basic
 - invoking methods 91
 - overview 91
- visual breakpoints
 - detecting 210
- Visual COBOL
 - about 179
 - supported versions 179

W

- web applications
 - creating tests 24
 - known issues 322
- Web applications
 - custom attributes 33, 84, 219, 260
 - supported attributes 219, 224
 - xBrowser test objects 189
- web pages
 - capturing, full page 53
- WebDriver
 - about 314
 - special keys, recording 316
- WebSync 329
- welcome 10
- Win32
 - priorLabel 184
- Windows
 - attribute types 168, 225
- Windows 8
 - limitations 220
- Windows 8.1
 - limitations 220
- Windows applications
 - creating tests 25
 - custom attributes 84
- Windows Forms
 - attribute types 163, 224
 - custom attributes 84
 - invoking methods 164
 - overview 163
- Windows Forms applications
 - custom attributes 164, 261
- Windows Presentation Foundation (WPF)
 - custom controls 170
 - exposing classes 174
 - invoking methods 170
 - locator attributes 168, 225
 - overview 167
 - WPFItemsControl class 169
- Windows-API
 - support 184
- WinForms applications
 - custom attributes 164, 261
- works order number 329
- WPF
 - classes, exposing 85
 - custom attributes 33
 - custom controls 170
 - exposing classes 174
 - invoking methods 170
 - locator attributes 168, 225
 - WPFItemsControl class 169
- WPF applications
 - custom attributes 84, 169, 261
- WPF locator attributes
 - identifying controls 168, 225
- writing TrueLogs
 - SilkTestCategory class 308
- wrong timestamps
 - logs, cross-browser tests 215

X

xBrowser

- Apple Safari 197
- attribute types 219, 224
- browser configuration settings 192
- Chrome for Android, setting 157
- class and style not in locators 217
- configuring locator generator 194
- cross-browser scripts 217
- current browser type, viewing 215
- custom attributes 84
- Dialog not recognized 214
- DomClick not working like Click 214
- exposing functionality 215
- FAQs 214
- FieldInputField.DomClick not opening dialog 214
- font type verification 215
- Google Chrome 201
- innerHTML 216
- innerText 216
- innerText not being used in locators 215
- Internet Explorer misplaces rectangles 216
- link.select focus issue 215

Microsoft Edge 208

mouse move preferences, setting 192

mouse move recording 216

Mozilla Firefox 205

navigating to new pages 216

object maps, using 267

object recognition 189

overview 186

page synchronization 190

playback, comparing API and native 191

recording an incorrect locator 216

recording locators 217

scrolling 214

test objects 189

textContent 216

wrong timestamps, logs 215

xBrowser testing

Apple Safari, limitations 199

current browser type, viewing 215

Microsoft Edge, limitations 208

XPath

creating query strings 262

troubleshooting 262