Borland
A MICRO FOCUS COMPANY

Silk Test 13.5

Silk Test Recorder Help

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

2012-09-19

Contents

Quick Start Tutorial for the Silk Test Recorder

This tutorial provides a step-by-step introduction to using the Silk Test Recorder to test a Web site. Silk Test Recorder uses *dynamic object recognition* to record and replay test cases that use XPath queries to find and identify objects.

Silk Test Recorder can record and replay tests for the following application types:

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

For the sake of simplicity, this guide assumes that you have installed the Silk Test Recorder and are using the Insurance Company sample Web site, *http://demo.borland.com/InsuranceWebExtJS/*.

Starting Silk Test Recorder

- Click **Start** > **Programs** > **Silk** > **Silk Test** > **Clients** > **Silk Test Recorder**. Silk Test Recorder opens and the Silk Test Open Agent icon appears in the system tray.

Recording a Test Case for the Insurance Company Web Site

Record a test case for the Insurance Company sample Web site to see how the Silk Test Recorder creates a test.

1. Perform one of the following steps:
   - Click in the toolbar.
   - Click **Record** > **Start Recording**.

   The **New Application Configuration** wizard opens.
2. Double-click **Web Site Test Configuration**. The **New Web Site Configuration** page opens.
3. From the **Browser Type** group, select **Internet Explorer**.

   You can use one of the other supported browser types to replay tests but not to record them.
4. Perform one of the following steps:
   - **Use existing browser** – Click this option button to use a browser that is already open when you configure the test. For example, if the Web page that you want to test is already displayed in a browser, you might want to use this option.

- **Start new browser** – Click this option button to start a new browser instance when you configure the test. Then, in the **Browse to URL** text box specify the Web page to open.

  For this tutorial, close all open browsers and then click **Start new browser** and specify *http://demo.borland.com/InsuranceWebExtJS/*.

5. Click **Finish**.

   If you have selected an existing instance of Google Chrome, Silk Test Recorder checks whether the automation support is included. If the automation support is not included, Silk Test Recorder informs you that Google Chrome has to be restarted.

   The website opens. Silk Test Recorder creates a base state and starts recording.
6. In the Insurance Company Web site, perform the following steps:
   a) From the **Select a Service or login** list box, select **Auto Quote**. The **Automobile Instant Quote** page opens.
   b) Type a zip code and email address in the appropriate text boxes, click an automobile type, and then click **Next**.
   c) Specify an age, click a gender and driving record type, and then click **Next**.
   d) Specify a year, make, and model, click the financial info type, and then click **Next**. A summary of the information you specified appears.
   e) Point to the **Zip Code** that you specified and press `Ctrl+Alt` to add a verification to the script.

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

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

   An action that corresponds with each step is recorded.
7. In the Recorder, perform one of the following steps:

   - Click  in the toolbar.
   - Click **Record** > **Stop Recording**.
8. Click **File** > **Save**.
   a) Navigate to the location in which you want to save the test.
   b) In the **File name** text box, type the name for the test and then click **Save**.
   For example, type `ZipTest`.

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

Replaying the Test Case for the Insurance Company Web Site

Replay a test to ensure that it works as expected.

1. On the main window, from the **Replay speed** list box, select the speed to use to replay the test.

   - **Fast** – This option is the fastest choice for test replay and the true speed at which scripts are executed. The other speeds use a delay mechanism, so you can see the test run.
   - **Medium** – In most cases, this option enables you to see each action.
   - **Slow** – To ensure that you see each action, choose this option.

   You can determine the speed of the test by checking the **Replay duration** time posted next the **Replay speed** list box.
2. To replay the entire test, click  in the toolbar or choose **Replay** > **Replay All**. Silk Test Recorder plays back the test.

3. If an error occurs, perform one of the following steps:
   - Click **Retry** to replay the current action.
   - Click **Stop** to end the test.
   - Click **Skip** to advance to the next action in the test.

Export the test to use it with Silk Test Classic, Silk4NET, or Silk4J. Or, modify the test to make changes if necessary.

Exporting a Test Case or Project

Use the following procedures to export a test case or project to Silk Test Classic, Silk4NET, or to Silk4J.

Exporting a Project to Silk4J

Export projects to use Silk4J as the primary GUI for tests and to organize test methods.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as Silk4J Project**. The **Export as Silk4J Project** page opens.
3. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click  and navigate to the folder that you want to use.
4. In the **Project name** text box, specify the project name.
   For example, type `Web Sample Project`.
5. In the **Package** text box, specify the package name.
   For example, type `com.example`.
6. In the **Test class** text box, specify the class name to which the test belongs.
   For example, type `AutoTests`.
7. In the **Test method** text box, specify a name for the test method.
   For example, type `TestAutoInput`.
8. From the **File encoding** list box, select the type of file encoding to use.
9. Click **Finish**.

   Silk Test Recorder creates a project and exports it to Silk4J.

Import the project with Silk4J. The new project contains a base state and test method and is ready for testing. For details about importing a project, refer to the *Silk4J User Guide*.

Exporting a Test Method to Silk4J

Export test methods to use Silk4J as the primary GUI for tests or to copy the JTF script to the clipboard.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as JTF Script**. The **Export to JTF** page opens.
3. From the **Export to** list box, select one of the following options:
   - **Clipboard** – Copies the JTF script to the clipboard. You might choose this option if you want to copy and paste the script into an existing JTF script.
   - **JTF Script** – Exports the script to Silk4J. You might choose this option if you want to create a new script or overwrite an existing JTF script.
4. In the **Test method** text box, specify a name for the test method.
   For example, type `TestAutoInput`.
5. In the **Package** text box, specify the package name.
   For example, type `com.example`.

6. In the **Test class** text box, specify the class name to which the test belongs.
   For example, type `AutoTests`.
7. In the **Source folder** text box, specify the location to which to export the test.

   Optional: Click … and navigate to the folder that you want to use.
8. From the **File encoding** list box, select the type of file encoding to use.
9. To include the base state in the exported script, check the **Use base state** check box.

   The base state makes sure that the application that you want to test is running and in the foreground. This ensures that tests will always start with the same application state, which makes them more reliable. In order to use the base state, it is necessary to specify what the main window looks like and how to launch the application that you want to test if it is not running. Creating a base state is optional. However, it is recommended as a best practice.

   If you export to JTF, a separate `silk4J.settings` file is created for the base state. If you export to the clipboard, a `Before` method includes the base state.
10. Click **Finish**.

    Silk Test Recorder creates a Java script and exports it to Silk4J or to the clipboard.

Exporting a Project to Silk Test Classic

Export projects to use Silk Test Classic as the primary GUI, group test cases into projects, or to share data with other products such as SilkCentral Test Manager.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as Silk Test Project**. The **Export as SilkTest Project** page opens.
3. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click … and navigate to the folder that you want to use.
4. In the **Project name** text box, specify the project name.
   For example, type `Web Sample Project`.
5. In the **4Test script** text box, specify the script file name.
   For example, type `AutoTests.t`.

   Optional: Click … and navigate to the folder that you want to use.
6. In the **Test case** text box, specify a name for the test case.
   For example, type `testAutoInput`.
7. To start Silk Test Classic after the test case is exported, check the **Open the exported project in Silk Test** check box.
8. Click **Finish**.

   Silk Test Recorder creates a project that includes a script that uses the 4Test language and a recovery file and exports the project to Silk Test Classic.

Use Silk Test Classic to work with the exported project. The new project contains a base state and test case and is ready for testing.

Exporting a Test Case to Silk Test Classic

Export test cases to use Silk Test Classic as the primary GUI or to copy the script to the clipboard.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as 4Test Script**. The **Export to 4Test Script** page opens.
3. From the **Export to** list box, select one of the following options:

   - **Clipboard** – Copies the script to the clipboard. You might choose this option if you want to copy and paste the script into an existing 4Test script.

- **4Test Script** – Exports the script to Silk Test Classic. You might choose this option if you want to create a new script or overwrite an existing script.

4. In the **Test case** text box, specify a name for the test case.
   For example, type `testAutoInput`.
5. In the **4Test script** text box, specify the script file name.
   For example, type `AutoTests.t`.

   Optional: Click  and navigate to the folder that you want to use.
6. To start Silk Test Classic after the test case is exported, check the **Open the exported script in Silk Test** check box.
7. Click **Finish**.

   Silk Test Recorder creates a script that uses the 4Test language and exports it to Silk Test Classic or the clipboard.

Exporting a Silk4NET Project to Visual Studio

Export a Silk4NET test and create a new Visual Studio project that uses Visual Basic .NET or C# as the primary programming language.

1. On the Silk Test Recorder menu bar, choose **File** > **Export** . The **Export** wizard opens.
2. Double-click **Export as Silk4NET Project**. The **Export as Silk4NET Project** page opens.
3. From the **Programming language** list box, specify whether the project uses Visual Basic .NET or C#.
4. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click  and navigate to the folder that you want to use.
5. In the **Project name** text box, specify the project name. For example, type `Visual Basic .NET Sample Project`.
6. In the **Namespace** text box, specify the container name for the project.
7. In the **Test class** text box, specify the class name to which the test belongs. For example, type `AutoTests`.
8. In the **Test method** text box, specify a name for the test method. For example, type `TestAutoInput`.
9. Click **Finish**. Silk Test Recorder creates a new project that includes the recorded test and exports the project to the specified location. From this location, you can open the project in Visual Studio.

Exporting Recorded Test Steps to Visual Studio

Export a test created by the Silk Test Recorder that you can add to a Visual Studio project. During the export, you can choose to create the test in either Visual Basic .NET or C#. Additionally, you can use the clipboard feature to copy and paste the recorded test steps into an existing Silk4NET test.

1. On the Silk Test Recorder menu bar, choose **File** > **Export** . The **Export** wizard opens.
2. Double-click **Export as NTF Script**. The **Export to NTF** page opens.
3. From the **Export to** list box, select one of the following options:

   - **Clipboard** – Copies the recorded test steps to the clipboard. Choose this option to copy and paste the recorded test steps into an existing Silk4NET test of a Visual Studio project.

     Note: You do not have to specify the source location or base state when selecting this option.
   - **NTF Script** – Exports the recorded test steps as a test you can add to an existing Visual Studio project. Choose this option if you want to create a new test or overwrite an existing test.
4. From the **Programming language** list box, specify whether the test uses Visual Basic .NET or C#.
5. In the **Test method** box, specify a name for the test method. For example, type `TestAutoInput`.

6. In the **Namespace** box, specify the container name for the test.
7. In the **Test class** box, specify the class name to which the test belongs. For example, type `AutoTests`.
8. In the **Source folder** box, specify the location to which to export the test.

   Optionally, click  and navigate to the folder that you want to use.
9. To include the base state in the exported test, check the **Use base state** check box.

   The base state makes sure that the application that you want to test is running and in the foreground. This ensures that tests will always start with the same application state, which makes them more reliable. In order to use the base state, it is necessary to specify what the main window looks like and how to launch the application that you want to test if it is not running. Creating a base state is optional. However, it is recommended as a best practice.
10. Click **Finish**. Recorder creates a Silk4NET test and exports it to the specified location or to the clipboard.
11. Add the exported Silk4NET project to your Visual Studio project. From the Visual Studio menu bar, choose **Project** > **Add Existing Item** , and then select the exported test.

Modifying the Test Case for the Insurance Company Web Site

After you record a test case, perform this step to manually modify the test. For example, you might add an additional action, change the order of the recorded steps, or change the parameters for a specific action.

1. Open the test that you want to modify.
2. Navigate to the Web page on which the test ended in the initial recording.
   For instance, if the existing test includes multiple Web pages, navigate to the last page in the test.

   In this case, the existing test recording ended on the **Automobile Instant Quote** summary page. We want to return to the **Home** page, which the base state will do automatically when we record.
3. To record the modifications, perform the following steps:
   a) Click  in the toolbar.
   b) From the **Select a Service or login** list box, select **Agent Lookup**.The **Find an Insurance Co. Agent** page opens.
   c) Click  in the toolbar.
4. To change an existing action, parameter, or locator string, perform the following steps:
   a) Click the row that you want to change in the Actions grid.
   For example, click the row that contains the **SetText** action for the e-mail address parameter.
   The **Action Details** tab shows the **Locator**, **Action**, and **Parameters**.
   b) To change the locator, type a string in the **Locator** text box.

   The locator string identifies the object that you want to test.
   c) Click **Validate Locator**. Silk Test Recorder validates the new locator. If the string is not valid or the object cannot be found, an error is displayed.
   d) To change the action, select an action from the **Action** list.
   e) To change the parameters, type a parameter value in the appropriate text box.
   For example, change the **text** parameter to `"tutorial@yourcompany.com"` for the **SetText** action.

   Any changes that you make display in the Actions grid immediately.
5. To manually add an action, perform the following steps:

   Typically, you record actions that you want to add to a test. However, you can also manually add an action by copying and pasting an existing action that you then modify.

a) Click an action in the **Actions** list.

Note: You can insert multiple actions by pressing `Ctrl` or `Shift` and then clicking the actions that you want to copy.

b) Perform one of the following steps:

- Click **Edit** > **Copy Selected Actions** and then choose **Edit** > **Paste Actions** .
- Press `Ctrl+C` and then press `Ctrl+V`.

A new action displays below the action that you selected.

c) In the Action Details view, modify the action to meet your needs by changing the locator, action name, or parameters as needed.

6. Click **File** > **Save**.

Replay the test to ensure that it works as expected.

Starting Silk Test Recorder

- Click **Start** > **Programs** > **Silk** > **Silk Test** > **Clients** > **Silk Test Recorder**. Silk Test Recorder opens and the Silk Test Open Agent icon appears in the system tray.

Replaying a Test Case

Replay a test to ensure that it works as expected.

1. Choose one of the following:

   - To replay the entire test, click  in the toolbar or choose **Replay** > **Replay All**.
   - To replay the selected action in the Actions grid and all actions following that action, choose **Replay** > **Replay From Selection**.
   - To replay the actions that you selected in the Actions grid, choose **Replay** > **Replay Selection**.

     To select more than one action, press `Ctrl` or `Shift` when you click the actions.

   Note: If you have another browser window open when you record the test, you must always have another browser window open when you replay the test. For instance, the locator string `.//BrowserApplication[3]` indicates that three browser windows were open when the test was recorded.

   Silk Test Recorder plays back the test.
2. If an error occurs, perform one of the following steps:

   - Click **Retry** to replay the current action.
   - Click **Stop** to end the test.
   - Click **Skip** to advance to the next action in the test.

Export the test to use it with Silk Test Classic, Silk4NET, or Silk4J. Or, modify the test to make changes if necessary.

Exporting a Test Case or Project

Use the following procedures to export a test case or project to Silk Test Classic, Silk4NET, or to Silk4J.

Exporting a Test Case to Silk Test Classic

Export test cases to use Silk Test Classic as the primary GUI or to copy the script to the clipboard.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as 4Test Script**. The **Export to 4Test Script** page opens.
3. From the **Export to** list box, select one of the following options:
   - **Clipboard** – Copies the script to the clipboard. You might choose this option if you want to copy and paste the script into an existing 4Test script.
   - **4Test Script** – Exports the script to Silk Test Classic. You might choose this option if you want to create a new script or overwrite an existing script.
4. In the **Test case** text box, specify a name for the test case.
   For example, type `testAutoInput`.
5. In the **4Test script** text box, specify the script file name.
   For example, type `AutoTests.t`.

   Optional: Click  and navigate to the folder that you want to use.
6. To start Silk Test Classic after the test case is exported, check the **Open the exported script in Silk Test** check box.
7. Click **Finish**.

   Silk Test Recorder creates a script that uses the 4Test language and exports it to Silk Test Classic or the clipboard.

Exporting a Test Method to Silk4J

Export test methods to use Silk4J as the primary GUI for tests or to copy the JTF script to the clipboard.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as JTF Script**. The **Export to JTF** page opens.
3. From the **Export to** list box, select one of the following options:
   - **Clipboard** – Copies the JTF script to the clipboard. You might choose this option if you want to copy and paste the script into an existing JTF script.
   - **JTF Script** – Exports the script to Silk4J. You might choose this option if you want to create a new script or overwrite an existing JTF script.
4. In the **Test method** text box, specify a name for the test method.
   For example, type `TestAutoInput`.
5. In the **Package** text box, specify the package name.
   For example, type `com.example`.
6. In the **Test class** text box, specify the class name to which the test belongs.
   For example, type `AutoTests`.
7. In the **Source folder** text box, specify the location to which to export the test.

   Optional: Click  and navigate to the folder that you want to use.

8. From the **File encoding** list box, select the type of file encoding to use.
9. To include the base state in the exported script, check the **Use base state** check box.

   The base state makes sure that the application that you want to test is running and in the foreground. This ensures that tests will always start with the same application state, which makes them more reliable. In order to use the base state, it is necessary to specify what the main window looks like and how to launch the application that you want to test if it is not running. Creating a base state is optional. However, it is recommended as a best practice.

   If you export to JTF, a separate `silk4J.settings` file is created for the base state. If you export to the clipboard, a `Before` method includes the base state.
10. Click **Finish**.

    Silk Test Recorder creates a Java script and exports it to Silk4J or to the clipboard.

Exporting Recorded Test Steps to Visual Studio

Export a test created by the Silk Test Recorder that you can add to a Visual Studio project. During the export, you can choose to create the test in either Visual Basic .NET or C#. Additionally, you can use the clipboard feature to copy and paste the recorded test steps into an existing Silk4NET test.

1. On the Silk Test Recorder menu bar, choose **File** > **Export** . The **Export** wizard opens.
2. Double-click **Export as NTF Script**. The **Export to NTF** page opens.
3. From the **Export to** list box, select one of the following options:
   - **Clipboard** – Copies the recorded test steps to the clipboard. Choose this option to copy and paste the recorded test steps into an existing Silk4NET test of a Visual Studio project.

     Note: You do not have to specify the source location or base state when selecting this option.
   - **NTF Script** – Exports the recorded test steps as a test you can add to an existing Visual Studio project. Choose this option if you want to create a new test or overwrite an existing test.
4. From the **Programming language** list box, specify whether the test uses Visual Basic .NET or C#.
5. In the **Test method** box, specify a name for the test method. For example, type `TestAutoInput`.
6. In the **Namespace** box, specify the container name for the test.
7. In the **Test class** box, specify the class name to which the test belongs. For example, type `AutoTests`.
8. In the **Source folder** box, specify the location to which to export the test.

   Optionally, click  and navigate to the folder that you want to use.
9. To include the base state in the exported test, check the **Use base state** check box.

   The base state makes sure that the application that you want to test is running and in the foreground. This ensures that tests will always start with the same application state, which makes them more reliable. In order to use the base state, it is necessary to specify what the main window looks like and how to launch the application that you want to test if it is not running. Creating a base state is optional. However, it is recommended as a best practice.
10. Click **Finish**. Recorder creates a Silk4NET test and exports it to the specified location or to the clipboard.
11. Add the exported Silk4NET project to your Visual Studio project. From the Visual Studio menu bar, choose **Project** > **Add Existing Item** , and then select the exported test.

Exporting a Project to Silk Test Classic

Export projects to use Silk Test Classic as the primary GUI, group test cases into projects, or to share data with other products such as SilkCentral Test Manager.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as Silk Test Project**. The **Export as SilkTest Project** page opens.
3. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click … and navigate to the folder that you want to use.
4. In the **Project name** text box, specify the project name.
   For example, type `Web Sample Project`.
5. In the **4Test script** text box, specify the script file name.
   For example, type `AutoTests.t`.

   Optional: Click … and navigate to the folder that you want to use.
6. In the **Test case** text box, specify a name for the test case.
   For example, type `testAutoInput`.
7. To start Silk Test Classic after the test case is exported, check the **Open the exported project in Silk Test** check box.
8. Click **Finish**.

   Silk Test Recorder creates a project that includes a script that uses the 4Test language and a recovery file and exports the project to Silk Test Classic.

Use Silk Test Classic to work with the exported project. The new project contains a base state and test case and is ready for testing.

Exporting a Project to Silk4J

Export projects to use Silk4J as the primary GUI for tests and to organize test methods.

1. Choose **File** > **Export**. The **Export** wizard opens.
2. Double-click **Export as Silk4J Project**. The **Export as Silk4J Project** page opens.
3. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click … and navigate to the folder that you want to use.
4. In the **Project name** text box, specify the project name.
   For example, type `Web Sample Project`.
5. In the **Package** text box, specify the package name.
   For example, type `com.example`.
6. In the **Test class** text box, specify the class name to which the test belongs.
   For example, type `AutoTests`.
7. In the **Test method** text box, specify a name for the test method.
   For example, type `TestAutoInput`.
8. From the **File encoding** list box, select the type of file encoding to use.
9. Click **Finish**.

   Silk Test Recorder creates a project and exports it to Silk4J.

Import the project with Silk4J. The new project contains a base state and test method and is ready for testing. For details about importing a project, refer to the *Silk4J User Guide*.

Exporting a Silk4NET Project to Visual Studio

Export a Silk4NET test and create a new Visual Studio project that uses Visual Basic .NET or C# as the primary programming language.

1. On the Silk Test Recorder menu bar, choose **File** > **Export** . The **Export** wizard opens.
2. Double-click **Export as Silk4NET Project**. The **Export as Silk4NET Project** page opens.
3. From the **Programming language** list box, specify whether the project uses Visual Basic .NET or C#.
4. In the **Project location** text box, specify the location to which to export the project.

   Optional: Click  and navigate to the folder that you want to use.
5. In the **Project name** text box, specify the project name. For example, type `Visual Basic .NET Sample Project`.
6. In the **Namespace** text box, specify the container name for the project.
7. In the **Test class** text box, specify the class name to which the test belongs. For example, type `AutoTests`.
8. In the **Test method** text box, specify a name for the test method. For example, type `TestAutoInput`.
9. Click **Finish**. Silk Test Recorder creates a new project that includes the recorded test and exports the project to the specified location. From this location, you can open the project in Visual Studio.

Setting Script Options

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

Setting Custom Attributes

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

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

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

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

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

1. Click **Settings** > **Script 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, Silk Test Recorder uses the default attributes for the application that you are testing.

   Separate attribute names with a comma.

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

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

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

5. Click **OK**.

Setting 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 **Settings** > **Script Options**. The **Script Options** dialog box opens.
2. Click the **Transparent Classes** tab.
3. In the **Custom WPF class names** grid, type the name of the class that you want to expose during recording and playback.

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

Setting Synchronization Options

Specify the synchronization and timeout values for Web applications.

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

1. Click **Settings** > **Script Options**. The **Script Options** dialog box opens.
2. Click the **Synchronization** tab.
3. To specify the synchronization algorithm for the ready state of a web application, from the **OPT_XBROWSER_SYNC_MODE** list box, choose an option.

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

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

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

For example, you might type:

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

Separate multiple entries with a comma.

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

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

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

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

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

Setting Replay Options

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

1. Click **Settings** > **Script Options**. The **Script Options** dialog box opens.
2. Click the **Replay** tab. The **Replay Options** page displays.
3. From the **OPT_REPLAY_MODE** list box, select one of the following options:

   - **Default** – Use this mode for the most reliable results. By default, each control uses either the mouse and keyboard (low level) or API (high level) modes. With the default mode, each control uses the best method for the control type.
   - **High level** – Use this mode to replay each control using the API.
   - **Low level** – Use this mode to replay each control using the mouse and keyboard.
4. To ensure that the object that you want to test is active, check the **OPT_ENSURE_ACTIVE_OBJDEF** check box.
5. Click **OK**.

Setting Windows Accessibility

For Win32 applications, specify whether Microsoft Accessibility (Accessibility) recognition should be enabled in addition to the normal Win32 control recognition.

1. Click **Settings** > **Script Options**. The **Script Options** dialog box opens.
2. Click the **Advanced** tab.

3. Check the **OPT_ENABLE_ACCESSIBILITY** check box to use Accessibility during recording.
4. Click **OK**.

Concepts

This section includes the conceptual overview topics for Silk Test Recorder.

Open Agent Port Numbers

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

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

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

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

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

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

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

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

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

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

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

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

   Port numbers can be any number from 1 to 65535.

   - `infoservice.default.port` – The default port where the information service runs. By default, this port is set to 22901.

- `infoservice.additional.ports` – A comma separated list of ports on which the information service runs if the default port is not available. By default, ports 2966, 11998, and 11999 are set as alternate ports.

3. Save the file as `infoservice.properties`.
4. If you are using Silk Test Recorder and you changed the information service port, specify the new port number in the Silk Test Recorder **Global Preferences** dialog box.
5. Restart the Silk Test Open Agent, Silk Test Classic, Silk Test Workbench, Silk Test Recorder, Silk4J, Silk4NET, and the application that you want to test.

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

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

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

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

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

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

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

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

   Port numbers can be any number from 1 to 65535.

   Port settings include:

   - `agent.vtadapter.port` – Controls communication between Silk Test Workbench and the Open Agent when running tests.
   - `agent.xpmodule.port` – Controls communication between Silk Test Classic and the Agent when running tests.
   - `agent.autcommunication.port` – Controls communication between the Open Agent and the application that you are testing.
   - `agent.rmi.port` – Controls communication with the Open Agent and Silk4J.
   - `agent.ntfadapter.port` – Controls communication with the Open Agent and Silk4NET.

   Note: The ports for Adobe Flex testing are not controlled by this configuration file. The assigned port for Flex application testing is 6000 and increases by 1 for each Flex application that is tested. You cannot configure the starting port for Flex testing.
3. Save the file as `agent.properties`.
4. Restart the Silk Test Open Agent, Silk Test Classic, Silk Test Workbench, Silk Test Recorder, Silk4J, Silk4NET, and the application that you want to test.

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

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

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

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

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

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

   Port numbers can be any number from 1 to 65535.

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

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

Supported Attribute Types

When a locator is constructed, the attribute type is automatically assigned based on the technology domain that your application uses. The attribute type and value determines how the locator identifies objects within your tests. If necessary, you can change the attribute type in one of the following ways:

- Manually typing another attribute type and value.
- Specifying another preference for the default attribute type by changing the **Preferred attribute list** values.

Attributes for Adobe Flex Applications

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

Supported attributes for Flex applications include:

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and `*`.

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

Attributes for Java AWT/Swing Applications

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

Supported attributes for Java AWT/Swing include:

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and `*`.

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.

Attributes for Java SWT Applications

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

Supported attributes for Java SWT include:

- caption
- all custom object definition attributes

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and `*`.

Locator Attributes for Identifying Rumba Controls

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

caption	The text that the control displays.
priorlabel	Since input fields on a form normally have a label explaining the purpose of the input, the intention of **priorlabel** is to identify the text input field, **RumbaTextField**, by the text of its adjacent label field, **RumbaLabel**. If no preceding label is found in the same line of the text field, or if the label at the right side is closer to the text field than the left one, a label on the right side of the text field is used.
StartRow	This attribute is not recorded, but you can manually add it to the locator. Use **StartRow** to identify the text input field, **RumbaTextField**, that starts at this row.
StartColumn	This attribute is not recorded, but you can manually add it to the locator. Use **StartColumn** to identify the text input field, **RumbaTextField**, that starts at this column.
All dynamic locator attributes.	For additional information on dynamic locator attributes, see *Dynamic Locator Attributes*.

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and `*`.

Attributes for SAP Applications

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

Supported attributes for SAP include:

- automationId
- caption

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and *.

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and *.

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

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

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

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

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

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

Attributes for Web Applications

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

Supported attributes for Web applications include:

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

Note: Empty spaces are handled differently by each browser. As a result, the `textContent` and `innerText` attributes have been normalized. Empty spaces are skipped or replaced by a single space if an empty space is followed by another empty space. Empty spaces are detected spaces, carriage returns, line feeds, and tabs. The matching of such values is normalized also. For example:

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

Uses the following locator:

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

Attributes for Windows API-based Client/Server Applications

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

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

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and `*`.

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.

Attributes for Windows Forms Applications

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

Supported attributes for Windows Forms applications include:

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and *.

Attributes for Windows Presentation Foundation (WPF) Applications

Supported attributes for WPF applications include:

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

Note: Attribute names are case sensitive. The locator attributes support the wildcards ? and *.

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

Object Recognition

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

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

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

During recording, Silk Test Recorder 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*, Silk Test Recorder 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>
```

Index

T

W

X