Borland
A MICRO FOCUS COMPANY

Silk Test 13.5

Migrating from the Classic Agent to the Open Agent

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

2012-09-19

Contents

Migrating from the Classic Agent to the Open Agent

This document provides an overview of the basic concepts of the Open Agent and explains the differences between the Classic Agent and the Open Agent. If you plan to migrate from testing using the Classic Agent to the Open Agent, review this information to learn how to migrate your existing assets, including window declarations and scripts.

Overview of Silk Test Agents

The Silk Test agent is the software process that translates the commands in your 4Test scripts 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.

Silk Test Classic provides two types of agents, the Open Agent and the Classic Agent. The agent that you assign to your project or script depends on the type of application that you are testing.

When you create a new project, Silk Test Classic automatically uses the agent that supports the type of application that you are testing. For instance, if you create an Adobe Flex or Windows API-based client/server project, Silk Test Classic uses the Open Agent.

When you open a project or script that was developed with the Classic Agent, Silk Test Classic automatically uses the Classic Agent. For instance, if you upgrade from Silk Test 2006 to Silk Test Classic, Silk Test Classic uses the Classic Agent for your existing projects.

For information about the supported technology domains for each agent, refer to *Testing in Your Environment*.

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

Overview of the Locator Keyword

Traditional Silk Test Classic scripts that use the Classic Agent use hierarchical object recognition. When you record a script that uses hierarchical object recognition, Silk Test Classic creates an include (.inc) file that contains window declarations and tags for the GUI objects that you are testing. Essentially, the INC file serves as a central global, repository of information about the application under test. It contains all the data structures that support your test cases and test scripts.

When you record a test case with the Open Agent, Silk Test Classic creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations. The locator is the actual name of the object, as opposed to the identifier, which is the logical name. Silk Test Classic uses the locator to identify objects in the application when executing test cases. Test cases never use the locator to refer to an object; they always use the identifier.

You can also manually create test cases that use dynamic object recognition without locator keywords. Dynamic object recognition uses a `Find` or `FindAll` function and an XPath query to locate the objects that you want to test. No include file, window declaration, or tags are required.

The advantages of using locators with an INC file include:

- You combine the advantages of INC files with the advantages of dynamic object recognition. For example, scripts can use window names in the same manner as traditional, Silk Test Classic tag-based scripts and leverage the power of XPath queries.

- Enhancing legacy INC files with locators facilitates a smooth transition from using hierarchical object recognition to new scripts that use dynamic object recognition. You use dynamic object recognition but your scripts look and feel like traditional, Silk Test Classic tag-based scripts that use hierarchical object recognition.
- You can use `AutoComplete` to assist in script creation. `AutoComplete` requires an INC file.

Syntax

The syntax for the locator keyword is:

```
[gui-specifier] locator locator-string
```

where locator-string is an XPath string. The XPath string is the same locator string that is used for the `Find` or `FindAll` functions.

Example

The following example shows a window declaration that uses locators:

```
window MainWin TestApplication
  locator "//MainWin[@caption='Test Application']"

  // The working directory of the application when it is invoked
  const sDir = "{SYS_GetEnv("SEGUE_HOME")}"

  // The command line used to invoke the application
  const sCmdLine = """{SYS_GetEnv("SEGUE_HOME")}testapp.exe"""

  Menu Control
    locator "//Menu[@caption='Control']"
    MenuItem CheckBox
      locator "//MenuItem[@caption='Check box']"
    MenuItem ComboBox
      locator "//MenuItem[@caption='Combo box']"
    MenuItem ListBox
      locator "//MenuItem[@caption='List box']"
    MenuItem PopupList
      locator "//MenuItem[@caption='Popup list']"
    MenuItem PushButton
      locator "//MenuItem[@caption='Push button']"
    MenuItem RadioButton
      locator "//MenuItem[@caption='Radio button']"
    MenuItem ListView
      locator "//MenuItem[@caption='List view']"
    MenuItem PageList
      locator "//MenuItem[@caption='Page list']"
    MenuItem UpDown
      locator "//MenuItem[@caption='Up-Down']"
    MenuItem TreeView
      locator "//MenuItem[@caption='Tree view']"
    MenuItem Textfield
      locator "//MenuItem[@caption='Textfield']"
    MenuItem StaticText
      locator "//MenuItem[@caption='Static text']"
    MenuItem TracKBar
      locator "//MenuItem[@caption='Track bar']"
    MenuItem ToolBar
      locator "//MenuItem[@caption='Tool bar']"
    MenuItem Scrollbar
      locator "//MenuItem[@caption='Scrollbar']"

  DialogBox CheckBox
    locator "//DialogBox[@caption='Check Box']"
    CheckBox TheCheckBox
      locator "//CheckBox[@caption='The check box']"
```

```
    PushButton Exit
      locator "//PushButton[@caption='Exit']"
```

For example, if the script uses a menu item like this:

```
TestApplication.Control.TreeView.Pick()
```

Then the menu item is resolved by using dynamic object recognition `Find` calls using XPath locator strings.

The above statement is equivalent to:

```
Desktop.Find(“//MainWin[@caption='Test Application']
  //Menu[@caption='Control']//MenuItem[@caption='Tree
view']”).Pick()
```

Locator String Syntax

For convenience, you can use shortened forms for the XPath locator strings. Silk Test Classic automatically expands the syntax to use full XPath strings when you run a script. You can omit:

- The hierarchy separator, “.//”. Silk Test Classic defaults to using “//”.
- The class name. Silk Test Classic defaults to the class name of the window that contains the locator.
- The surrounding square brackets of the attributes,"[ ]".
- The “@caption=’” if the xPath string refers to the caption.

The following locators are equivalent:

```
Menu Control
  //locator "//Menu[@caption='Control']"
  //locator "Menu[@caption='Control']"
  //locator "[@caption='Control']"
  //locator "@caption='Control'"
  locator "Control"
```

You can use shortened forms for the XPath locator strings only when you use an INC file. For scripts that use dynamic object recognition without an INC file, you must use full XPath strings.

Window Hierarchies

You can create window hierarchies without locator strings. In the following example, the “Menu Control” acts only as a logical hierarchy, used to provide the INC file with more structure. “Menu Control” does not contribute to finding the elements further down the hierarchy.

```
window MainWin TestApplication
  locator "//MainWin[@caption='Test Application']"
  Menu Control
    MenuItem TreeView
      locator "//MenuItem[@caption='Tree view']"
```

In this case, the statement:

```
TestApplication.Control.TreeView.Pick()
```

is equivalent to:

```
Desktop.Find(“.//MainWin[@caption='Test Application']
  //MenuItem[@caption='Tree view']”).Pick()
```

Including Locators and Tags in the Same Window Declaration

You can include locators and tags in the same window declaration. For example:

```
window MainWin TestApplication
  locator "Test Application"
  tag "Test Application"
  Menu Control
```

```
    tag "Control"
    MenuItem TreeView
      locator "Tree view"
      tag "Tree view"
```

The following rules determine if locators or tags are used for resolving the window at runtime:

- When replaying a script on the Classic Agent, only tags are used. Locators are never used with the Classic Agent.
- When replaying a script on the Open Agent, locators are used if a locator is present for the bottom-most window in the hierarchy and the **Prefer Locator** check box is checked in the **General Options** dialog box. By default, the **Prefer Locator** check box is checked.
- If the top-most window specifies a locator, the Open Agent is used.

In the preceding example, `TestApplication.Control.Open()` uses tags for resolving because the Menu Control does not specify a locator.

`TestApplication.Control.TreeView.Pick()` uses locators for resolving because the MenuItem TreeView specifies a locator and the **Prefer Locator** check box is checked.

Expressions

You can use expressions in locators. For example, you can specify:

```
STRING getSWTVersion()
  return SYS_GETENV("SWT_VERSION")
window Shell SwtTestApplication
  locator "SWT {getSWTVersion()} Test Application"
```

Comparing the Locator Keyword to the Tag Keyword

The syntax of locators is identical to the syntax of the tag keyword.

The overall rules for locators are the same as for tags. There can be only one locator per window, except for different gui-specifiers, in this case there can be only one locator per gui-specifier.

You can use expressions in locators and tags.

The locator keyword requires a script that uses the Open Agent while the tag keyword requires a script that uses the Classic Agent.

Hierarchical Object Recognition

When you record window declarations, Silk Test Classic records descriptions based on hierarchical object recognition of the GUI objects in your application. Silk Test Classic stores the declarations in an include file (*.inc). When you record or replay a test case, Silk Test Classic references the declarations in the include file to identify the objects named in your test scripts.

Using hierarchical object recognition compared to using dynamic object recognition

Use hierarchical object recognition to test applications that require the Classic Agent. Dynamic object recognition requires the Open Agent.

Alternatively, you can combine the advantages of INC files with the advantages of dynamic object recognition by including locator keywords in INC files. Enhancing INC files with locators facilitates a smooth transition from using hierarchical object recognition to new scripts that use dynamic object recognition. With locators, you use dynamic object recognition but your scripts look and feel like traditional, Silk Test Classic tag-based scripts that use hierarchical object recognition.

You can create tests for both dynamic and hierarchical object recognition in your test environment. You can use both recognition methods within a single test case if necessary. Use the method best suited to meet your test requirements.

Open Agent Example

For example, if you record a test to open the **New Window** dialog box by clicking **File** > **New** > **Window** in the SWT sample application, Silk Test Classic performs the following tasks:

- Records the following test:

```
testcase Test1 ()
  recording
    SwtTestApplication.WindowMenuItem.Pick()
```

- Creates window declarations in the include file for Window menu item. For example:

```
window Shell SwtTestApplication
  locator "/Shell[@caption='Swt Test Application']"
  MenuItem WindowMenuItem
    locator "//MenuItem[@caption='Window']"
```

Classic Agent Example

For example, if you record a test to open the **New Window** dialog box by clicking **File** > **New** > **Window** in a sample application, Silk Test Classic performs the following tasks:

- Records the following test:

```
testcase Test1 ()
  recording
    SwtTestApplication.File.New.xWindow.Pick()
```

- Creates window declarations in the include file for File menu, New menu item, and xWindow menu item. For example:

```
Menu File
  tag "File"
  MenuItem New
    tag "New.."
    MenuItem xWindow
      tag "Window"
```

XPath Basic Concepts

Silk Test Classic 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.

Silk Test Classic provides an alternative to using `Find` or `FindAll` functions in scripts that use XPath queries. You can use locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Supported XPath Subset

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

To create tests that use dynamic object recognition, you must use the Open Agent.

The following table lists the constructs that Silk Test Classic supports.

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

The following table lists the XPath constructs that Silk Test Classic does not support.

Unsupported XPath Construct	Example
Comparing two attributes with each other.	`PushButton[@caption = @windowid]`
An attribute name on the right side is not supported. An attribute name must be on the left side.	`PushButton['abc' = @caption]`
Combining multiple XPath expressions with 'and' or 'or'.	`PushButton [@caption = 'abc'] or .// Checkbox`
More than one set of attribute brackets.	`PushButton[@caption = 'abc] [@windowid = '123']` Use `PushButton [@caption = 'abc and @windowid = '123']` instead.
More than one set of index brackets.	`PushButton[1][2]`
Any construct that does not explicitly specify a class or the class wildcard, such as including a wildcard as part of a class name.	`//[@caption = 'abc']` Use `//*[@caption = 'abc']` instead. "//*Button[@caption='abc']"

Recording Locators Using the Locator Spy

This functionality is supported only if you are using the Open Agent.

Capture a locator using the **Locator Spy** and copy the locator to the test case or to the Clipboard.

1. Configure the application to set up the technology domain and base state that your application requires.
2. Click **File** > **New**. The **New File** dialog box opens.
3. Select **4Test script** and then click **OK**. A new 4Test Script window opens.
4. Click **Record** > **Window Locators**. The **Locator Spy** opens.
5. Position the mouse over the object that you want to record. The related locator XPath query string shows in the **Selected Locator** text box. The **Locator Details** section lists the hierarchy of objects for the locator that displays in the text box.
6. Perform one of the following steps:
   - Press **Ctrl+Alt** to capture the object with the default Record Break key sequence.
   - Press **Ctrl+Shift** to capture the object if you specified the alternative Record Break key sequence on the **General Recording Options** page of the **Recording Options** dialog box.

     Note: For SAP applications, you must set **Ctrl+Shift** as the shortcut key combination to use to pause recording. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.
   - Click **Stop Recording Locator**.
   - If you use Picking mode, click the object that you want to record and press the Record Break keys.

   Note: Silk Test Classic does not verify whether the locator string is unique. We recommend that you ensure that the string is unique. Otherwise additional objects might be found when you run the test. Furthermore, you might want to exclude some of the attributes that Silk Test Classic identifies because the string will work without them.
7. To refine the locator, in the **Locator Details** table, click **Show additional locator attributes**, right-click an object and then choose **Expand subtree**. The objects display and any related attributes display in the **Locator Attribute** table.
8. *Optional:* You can replace a recorded locator attribute with another locator attribute from the **Locator Details** table.

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

```
/Window[@caption='MyApp']//Control[@id='table1']
```

If you have a caption `Files` listed in the **Locator Details** table, you can manually change the locator to the following:

```
/Window[@caption='MyApp']//Control[@caption='Files']
```

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

9. Copy the locator to the test case or to the Clipboard.
10. Click **Close**.

Recording Window Declarations that Include Locator Keywords

A window declaration specifies a cross-platform, logical name for a GUI object, called the identifier, and maps the identifier to the object's actual name, called the tag or locator. You can use locator keywords, rather than tags, to create scripts that use dynamic object recognition and window declarations. Or, you can include locators and tags in the same window declaration.

To record window declarations that include locator keywords, you must use the Open Agent.

To record window declarations using the Locator Spy:

1. Configure the application to set up the technology domain and base state that your application requires.
2. Click **Record** > **Window Locators**. The **Locator Spy** opens.
3. Position the mouse over the object that you want to record and perform one of the following steps:
   - Press **Ctrl+Alt** to capture the object hierarchy with the default Record Break key sequence.
   - Press **Ctrl+Shift** to capture the object hierarchy if you specified the alternative Record Break key sequence on the **General Recording Options** page of the **Recording Options** dialog box.

     Note: For SAP applications, you must set **Ctrl+Shift** as the shortcut key combination. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.
   - If you use Picking mode, click the object that you want to record and press the Record Break keys.
4. Click **Stop Recording Locator**.

   The **Locator** text box displays the XPath query string for the object on which the mouse rests. The **Locator Details** section lists the hierarchy of objects for the locator that displays in the text box. The hierarchy listed in the **Locator Details** section is what will be included in the INC file.
5. To refine the locator, in the **Locator Details** table, click **Show additional locator attributes**, right-click an object and then choose **Expand subtree**. The objects display and any related attributes display in the **Locator Attribute** table.
6. To replace the hierarchy that you recorded, select the locator that you want to use as the parent in the **Locator Details** table. The new locator displays in the **Locator** text box.
7. Perform one of the following steps:
   - To add the window declarations to the INC file for the project, position your cursor where you want to add the window declarations in the INC file, and then click **Paste Hierarchy to Editor**.
   - To copy the window declarations to the Clipboard, click **Copy Hierarchy to Clipboard** and then paste the window declarations into a different editing window or into the current window at the location of your choice.
8. Click **Close**.

Differences Between the Classic Agent and the Open Agent

This section describes the key differences between the Classic Agent and the Open Agent.

Differences for Agent Options Between the Classic Agent and the Open Agent

Before you migrate existing Classic Agent scripts to the Open Agent, review the Agent Options listed below to determine if any additional action is required to facilitate the migration.

Agent Option	Action for Open Agent
OPT_AGENT_CLICKS_ONLY	Option not needed.  **Note:** Use `OPT_REPLAY_MODE` for switching between high-level (API) clicks and low-level clicks.
OPT_CLOSE_MENU_NAME	Not supported by Open Agent.
OPT_COMPATIBLE_TAGS	Option not needed.
OPT_COMPRESS_WHITESPACE	Not supported by Open Agent.
OPT_DROPDOWN_PICK_BEFORE_GET	Option not needed. The Open Agent performs this action by default during replay.
OPT_EXTENSIONS	Option not needed.
OPT_GET_MULTITEXT_KEEP_EMPTY_LINES	Not supported by Open Agent.
OPT_KEYBOARD_LAYOUT	Not supported by Open Agent.
OPT_MENU_INVOKE_POPUP	No action. Pop-up menu handling using the Open Agent does not need such an option.
OPT_MENU_PICK_BEFORE_GET	Option not needed.
OPT_NO_ICONIC_MESSAGE_BOXES	Option not needed.
OPT_PLAY_MODE	Option not needed.
OPT_RADIO_LIST	Open Agent always sees `RadioList` items as individual objects.
OPT_REL1_CLASS_LIBRARY	Obsolete option.
OPT_REQUIRE_ACTIVE	Use the option `OPT_ENSURE_ACTIVE` instead.
OPT_SCROLL_INTO_VIEW	Option not needed. Open Agent only requires scrolling into view for low-level replay. By default, high-level replay is used, so no scrolling needs to be performed. However, `CaptureBitmap` never scrolls an object into view.
OPT_SET_TARGET_MACHINE	Option not needed.
OPT_SHOW_OUT_OF_VIEW	Option not needed. Out-of-view objects are always recognized.
OPT_TEXT_NEW_LINE	Option not needed. The Open Agent always uses **Enter** to type a new line.
OPT_TRANSLATE_TABLE	Not supported by Open Agent.

Agent Option	Action for Open Agent
OPT_TRAP_FAULTS	Fault trap is no longer active.
OPT_TRAP_FAULTS_FLAGS	Fault trap is no longer active.
OPT_TRIM_ITEM_SPACE	Option not needed. If required, use a * wildcard instead.
OPT_USE_ANSICALL	Not supported by Open Agent.
OPT_USE_SILKBEAN	SilkBean is not supported on the Open Agent.
OPT_VERIFY_APPREADY	Option not needed. The Open Agent performs this action by default.
OPT_VERIFY_CLOSED	Option not needed. The Open Agent performs this action by default.
OPT_VERIFY_COORD	Option not needed. The Open Agent does not typically check for native input in order to allow clicking outside of an object.
OPT_VERIFY_CTRLTYPE	Option not needed.
OPT_VERIFY_EXPOSED	Option not needed. The Open Agent performs this action when it sets a window to active. `OPT_ENSURE_ACTIVE_OBJECT_DEF` should yield the same result.
OPT_VERIFY_RESPONDING	Option not needed.
OPT_WINDOW_MOVE_TOLERANCE	Option not needed.

Differences in Object Recognition Between the Classic Agent and the Open Agent

When recording and executing test cases, the Classic Agent uses the keywords tag or multitag in a window declaration to uniquely identify an object in the test application. The tag is the actual name, as opposed to the identifier, which is the logical name.

When using the Open Agent, you typically use dynamic object recognition with a `Find` or `FindAll` function and an XPath query to locate objects in your test application. To make calls that use window declarations using the Open Agent, you must use the keyword locator in your window declarations. Similar to the tag or multitag keyword, the locator is the actual name, as opposed to the identifier, which is the logical name. This similarity facilitates a smooth transition of legacy window declarations, which use the Classic Agent, to dynamic object recognition, which leverages the Open Agent.

The following sections explain how to migrate the different tag types to valid locator strings.

Caption

Classic Agent	tag “<caption string>”
Open Agent	locator “//<class name>[@caption=’<caption string>’]”

Note: For convenience, you can use shortened forms for the XPath locator strings. Silk Test Classic automatically expands the syntax to use full XPath strings when you run a script.

You can omit:

- The hierarchy separator, “.//”. Silk Test Classic defaults to “//”.
- The class name. Silk Test Classic defaults to the class name of the window that contains the locator.
- The surrounding square brackets of the attributes, “[ ]”.
- The “@caption=” if the XPath string refers to the caption.

Note: Classic Agent removes ellipses (…) and ampersands (&) from captions. Open Agent removes ampersands, but not ellipses.

Example

Classic Agent:

```
CheckBox CaseSensitive
  tag “Case sensitive”
```

Open Agent:

```
CheckBox CaseSensitive
  locator “//CheckBox[@caption='Case sensitive']”
```

Or, if using the shortened form:

```
CheckBox CaseSensitive
  locator “Case sensitive”
```

Prior text

Classic Agent	tag “^Find What:”
Open Agent	locator “//<class name>[@priorlabel=’Find What:’]”

Note: Only available for Windows API-based and Java Swing applications. For other technology domains, use the **Locator Spy** to find an alternative locator.

Index

Classic Agent	tag “#1”
Open Agent	Record window locators for the test application. The Classic Agent creates index values based on the position of controls, while the Open Agent uses the controls in the order provided by the operating system. As a result, you must record window locators to identify the current index value for controls in the test application.

Window ID

Classic Agent	tag “$1041”
Open Agent	locator “//<class name>[@windowid=’1041’]”

Location

Classic Agent	tag “@(57,75)”
Open Agent	not supported

Note: If you have location tags in your window declarations, use the **Locator Spy** to find an alternative locator.

Multitag

Classic Agent	multitag “Case sensitive” “$1011”
Open Agent	locator “//CheckBox[@caption=’Case sensitive’ or @windowid=’1011’]” ‘parent’ statement

No changes needed. Multitag works the same way for the Open Agent.

Differences in the Classes Supported by the Open Agent and the Classic Agent

The Classic Agent and the Open Agent differ slightly in the types of classes that they support. These differences are important if you want to manually script your test cases. Or, if you are testing a single test environment with both the Classic Agent and the Open Agent. Otherwise, the Open Agent provides the majority of the same record capabilities as the Classic Agent and the same replay capabilities.

Windows-based applications

Both Agents support testing Windows API-based client/server applications. The Open Agent classes, functions, and properties differ slightly from those supported on the Classic Agent for Windows API-based client/server applications.

Classic Agent	Open Agent
AnyWin	AnyWin
AgentClass (Agent)	AgentClass (Agent)
CheckBox	CheckBox
ChildWin	<no corresponding class>
ClipboardClass (Clipboard)	ClipboardClass (Clipboard)
ComboBox	ComboBox
Control	Control
CursorClass (Cursor)	CursorClass (Cursor)
CustomWin	CustomWin
DefinedWin	<no corresponding class>
DesktopWin (Desktop)	DesktopWin (Desktop)
DialogBox	DialogBox
DynamicText	<no corresponding class>
Header	HeaderEx
ListBox	ListBox
ListView	ListViewEx
MainWin	MainWin
Menu	Menu
MenuItem	MenuItem
MessageBoxClass	<no corresponding class>
MoveableWin	MoveableWin
PageList	PageList
PopupList	ComboBox
PopupMenu	<no corresponding class>
PopupStart	<no corresponding class>

Classic Agent	Open Agent
PopupSelect	<no corresponding class>
PushButton	PushButton
RadioButton	**Note:** Items in Radiolists are recognized as RadioButtons on the CA. OA only identifies all of those buttons as RadioList.
RadioList	RadioList
Scale	Scale
ScrollBar	ScrollBar, VerticalScrollBar, HorizontalScrollBar
StaticText	StaticText
StatusBar	StatusBar
SysMenu	<no corresponding class>
Table	TableEx
TaskbarWin (Taskbar)	<no corresponding class>
TextField	TextField
ToolBar	ToolBar Additionally: PushToolItem, CheckBoxToolItem
TreeView, TreeViewEx	TreeView
UpDown	UpDownEx

The following core classes are supported on the Open Agent only:

- CheckBoxToolItem
- DropDownToolItem
- Group
- Item
- Link
- MonthCalendar
- Pager
- PushToolItem
- RadioListToolItem
- ToggleButton
- ToolItem

Web-based Applications

Both Agents support testing Web-based applications. The Open Agent classes, functions, and properties differ slightly from those supported on the Classic Agent for Windows API-based client/server applications.

Classic Agent	Open Agent
Browser	BrowserApplication
BrowserChild	BrowserWindow
HtmlCheckBox	DomCheckBox
HtmlColumn	<no corresponding class>

Classic Agent	Open Agent
HtmlComboBox	<no corresponding class>
HtmlForm	DomForm
HtmlHeading	<no corresponding class>
HtmlHidden	<no corresponding class>
HtmlImage	<no corresponding class>
HtmlLink	DomLink
HtmlList	<no corresponding class>
HtmlListBox	DomListBox
HtmlMarquee	<no corresponding class>
HtmlMeta	<no corresponding class>
HtmlPopupList	DomListBox
HtmlPushButton	DomButton
HtmlRadioButton	DomRadioButton
HtmlRadioList	<no corresponding class>
HtmlTable	DomTable
HtmlText	<no corresponding class>
HtmlTextField	DomTextField
XmlNode	<no corresponding class>
Xul* Controls	<no corresponding class>

Java AWT/Swing Applications

Both Agents support testing Java AWT/Swing applications. The Open Agent classes, functions, and properties differ slightly from those supported on the Classic Agent for Windows API-based client/server applications.

Classic Agent	Open Agent
JavaApplet	AppletContainer
JavaDialogBox	AWTDialog, JDialog
JavaMainWin	AWTFrame, JFrame
JavaAwtCheckBox	AWTCheckBox
JavaAwtListBox	AWTList
JavaAwtPopupList	AWTChoice
JavaAwtPopupMenu	<no corresponding class>
JavaAwtPushButton	AWTPushButton
JavaAwtRadioButton	AWTRadioButton
JavaAwtRadioList	<no corresponding class>

Classic Agent	Open Agent
JavaAwtScrollBar	AWTScrollBar
JavaAwtStaticText	AWTLabel
JavaAwtTextField	AWTTextField, AWTTextArea
JavaJFCCheckBox	JCheckBox
JavaJFCCheckBoxMenuItem	JCheckBoxMenuItem
JavaJFCChildWin	<no corresponding class>
JavaJFCComboBox	JComboBox
JavaJFCImage	<no corresponding class>
JavaJFCListBox	JList
JavaJFCMenu	JMenu
JavaJFCMenuItem	JMenuItem
JavaJFCPageList	JTabbedPane
JavaJFCPopupList	JList
JavaJFCPopupMenu	JPopupMenu
JavaJFCProgressBar	JProgressBar
JavaJFCPushButton	JButton
JavaJFCRadioButton	JRadioButton
JavaJFCRadioButtonMenuItem	JRadioButtonMenuItem
JavaJFCRadioList	<no corresponding class>
JavaJFCScale	JSlider
JavaJFCScrollBar	JScrollBar, JHorizontalScrollBar, JVerticalScrollBar
JavaJFCSeparator	JComponent
JavaJFCStaticText	JLabel
JavaJFCTable	JTable
JavaJFCTextField	JTextField, JTextArea
JavaJFCToggleButton	JToggleButton
JavaJFCToolBar	JToolBar
JavaJFCTreeView	JTree
JavaJFCUpDown	JSpinner

Java SWT/RCP Applications

Only the Open Agent supports testing Java SWT/RCP-based applications. For a list of the classes, see *Supported SWT Widgets for the Open Agent*.

Overview of the Methods Supported by the Silk Test Classic Agents

The `winclass.inc` file includes information about which methods are supported for each Silk Test Classic Agent. The following 4Test keywords indicate Agent support:

supported_ca	Supported on the Classic Agent only.
supported_oa	Supported on the Open Agent only.

Standard 4Test methods, such as `AnyWin::GetCaption()`, can be marked with one of the preceding keywords. A method that is marked with the supported_ca or supported_oa keyword can only be executed successfully on the corresponding Agent. Methods that do not have a keyword applied will run on both Agents.

To find out which methods are supported on each Agent, open the .inc file, for instance `winclass.inc`, and verify whether the supported_ca or supported_oa keyword is applied to it.

Classic Agent

Certain functions and methods run on the Classic Agent only. When these are recorded and replayed, they default to the Classic Agent automatically. You can use these in an environment that uses the Open Agent. Silk Test Classic automatically uses the appropriate Agent. The functions and methods include:

- C data types for use in calling functions in DLLs.
- `ClipboardClass` methods.
- `CursorClass` methods.
- Certain SYS functions.

Index