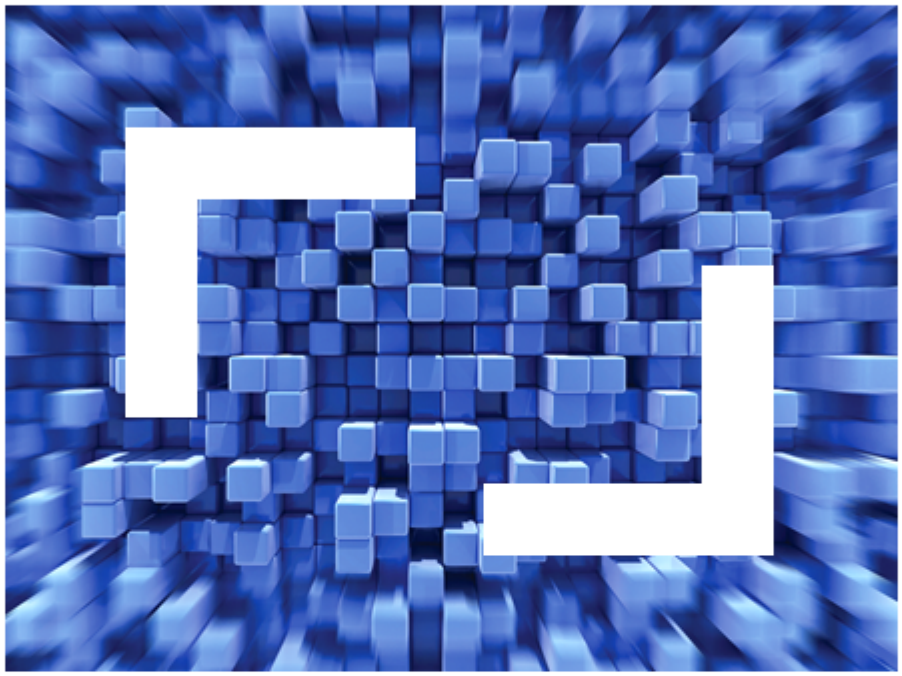


SilkTest 2011

SilkTest Classic Help

Borland®
(A MICRO FOCUS COMPANY)

MICRO® FOCUS

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

2011-10-10

Contents

Welcome

Welcome to SilkTest Classic.

SilkTest Classic is the industry's leading functional testing product for e-business applications, whether Web, Java, or traditional client-server based.

- For information on how to use this Help system, see Help on Help.
- You can find the entire documentation set for SilkTest Classic on *http://supportline.microfocus.com*.

Note: If your Internet access is limited by network security, or if your computer is protected by a personal firewall, the web-based links in this Help system might not function properly.

Not all features described in this Help system are available in all editions of the product.

What's New in SilkTest Classic

SilkTest Classic supports the following new features:

Microsoft Silverlight Support

SilkTest includes record and playback support for applications developed with Microsoft Silverlight.

Micro Focus Rumba Support

SilkTest includes record and playback support for applications developed with Micro Focus Rumba.

64 Bit Support for Java Applications

SilkTest includes support for testing 64-bit Java applications developed using the following Java toolkits:

- AWT
- Swing
- Standard Widget Toolkit (SWT)

Internet Explorer 9 Support

SilkTest includes record and playback support for applications running in an Internet Explorer 9 Web browser.

Mozilla Firefox 5 and Mozilla Firefox 6 Support

SilkTest includes playback support for applications running in a Mozilla Firefox 5 or Mozilla Firefox 6 Web browser.

User Account Control

SilkTest now enables you to test applications with UAC enabled. You only need to disable UAC when you are testing applications with SilkTest Classic and the Classic Agent.

Note: If you want to use SilkTest with UAC enabled, we recommend you not to install SilkTest into the `Program Files` directory, because SilkTest requires writing permissions to the install directory. However, if you decide to install SilkTest into the `Program Files` directory, you need administrator permissions to configure the database for SilkTest.

Adobe Flex 4.x Support

In addition to the new Flex 4.x controls, SilkTest also supports multiple application domains in Flex 3.x and 4.x applications, which enables you to test sub-applications.

Contacting Micro Focus

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

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

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

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

Information Needed by Micro Focus SupportLine

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

- The name and version number of all products that you think might be causing an issue.
- Your computer make and model.
- System information such as operating system name and version, processors, and memory details.
- Any detailed description of the issue, including steps to reproduce the issue.
- Exact wording of any error messages involved.
- Your Software Support Identification Number (SHIN) if you have one (not used in all countries).
- Your serial number.

To find out these numbers, look .

Help on Help

This section includes information about:

- SilkTest Help
- Typographic Conventions Used in the Help

SilkTest Help

SilkTest Help includes conceptual overviews and procedural how-to topics. These topic types allow you to navigate from general to more specific information as needed.

Conceptual Overviews

The conceptual overviews provide information about product architecture, components, and other information you need to work with SilkTest. At the end of most of the overviews, you will find links to related, more detailed information.

How-To Procedures

The how-to procedures provide step-by-step instructions. At the end of most of the procedures, you will find links to related procedures. Additionally, most of the conceptual overviews provide links to the pertinent procedures.

Typographic Conventions Used in the Help

The following typographic conventions are used in the SilkTest online Help.

Monospace type	Source code and text that you must type.
Boldface	References to dialog boxes and other user interface elements.
Italics	Identifiers, such as variables. Italicized text is also used to emphasize new terms.

Getting Started

Basic Workflow

The **Basic workflow** bar guides you through the process of creating a test case.

To create and execute a test case, click each icon in the workflow bar to perform the relevant procedures. The procedures and the appearance of the workflow bar differ depending on whether your test uses the Open Agent or Classic Agent.

The **Basic workflow** is on by default. You can turn it on and off by choosing **Workflows** > **Basic**. If your test uses both the Open Agent and Classic Agent, change the default agent to change the **Basic workflow** bar.

When you use the Open Agent, the **Basic workflow** uses dynamic object recognition to record and replay test cases that use XPath queries to find and identify objects.

When you use the Classic Agent, the **Basic workflow** uses hierarchical object recognition to record and replay test cases that use window declarations to find and identify objects.

Open Agent

This section includes the tasks that you can perform in the **Basic Workflow** toolbar, when you are using the Open Agent.

Creating a New Project

You can create a new project and add the appropriate files to the project, or you can have SilkTest Classic automatically create a new project from an existing file.

Since each project is a unique testing environment, by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as an options set by opening SilkTest Classic and clicking **Options** > **Save New Options Set**. You can add the options set to your project.

To create a new project:

1. In SilkTest Classic, click **File** > **New Project**, or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click the type of project that you want to test.

   The type of project that you select determines the default Agent. For instance, if you specify that you want to create an Adobe Flex project, the Open Agent is automatically set as the default agent. SilkTest Classic uses the default agent when recording a test case, enabling extensions, or configuring an application.
3. Click **OK**.
4. On the **Create Project** dialog box, type the **Project Name** and **Description**.
5. Click **OK** to save your project in the default location, `<SilkTest Classic installation directory>\Project`.

   To save your project in a different location, click **Browse** and specify the folder in which you want to save your project.

   SilkTest Classic creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` files to this location and copies the extension .ini files, which are `appexpex.ini`,

`axext.ini`, `domex.ini`, and `javaex.ini` to a `projectname\extend` subdirectory. SilkTest Classic then creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources associated with this project.

6. Perform one of the following steps:
   - If your test uses the Open Agent, configure the application to set up the test environment.
   - If your test uses the Classic Agent, enable the appropriate extensions to test your application.

Configuring Applications

When you configure an application, SilkTest automatically creates a base state for the application. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution.

SilkTest has slightly different procedures depending on whether you are configuring a Web application or an application that does not use a Web browser, such as a Windows application, Java SWT application, or an Adobe Flex application.

Configuring Web Applications

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a testcase. If you are testing a Flex application, use this configuration.

1. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
2. Double-click **Web Site Test Configuration**.
   The **New Web Site Configuration** page opens.
3. From the **Browser Type** list box, select **Internet Explorer**.

   You can use Firefox to replay tests but not to record them.
4. Perform one of the following steps:
   - Click **Use existing browser** to use a browser window that is already open to configure the test. For example, if the Web page that you want to test is already displayed in the browser window, you might want to use this option.
   - Click **Start new browser** to start a new browser instance to configure the test. Then, in the **Browse to URL** text box specify the Web page to open.
5. Click **Finish**.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
6. Navigate to the location in which you want to save the frame file.
7. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

   SilkTest automatically creates a base state for the application. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box. For instance, if you configure an application that uses an Internet Explorer or Firefox browser, SilkTest adds the `xBrowser.inc` file to the **Runtime Options** dialog box.

   SilkTest opens the Web page.
8. Record the test case whenever you are ready.

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.
7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

   Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.
8. Click Finish.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

SilkTest opens the include file.

11. Record the test case whenever you are ready.

Recording Test Cases With the Open Agent

When you record a test case with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations. With this approach, 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, SilkTest tag-based scripts and leverage the power of XPath queries.

1. Click **Record Testcase** on the basic workflow bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
   The **Record Testcase** dialog box opens.
2. Type the name of your test case in the **Testcase name** text box.

   Test case names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
3. From the **Application State** list box, select **DefaultBaseState** to have the built-in recovery system restore the default base state before the test case begins executing.

   - If you choose DefaultBaseState as the application state, the test case is recorded in the script file as `testcase testcase_name ()`.
   - If you chose another application state, the test case is recorded as `testcase testcase_name () appstate appstate_name`.
4. If you do not want SilkTest to display the status window during playback when driving the application to the specified base state, uncheck the **Show AppState status window** check box.

   Typically, you check this check box. However, in some circumstances it is necessary to hide the status window. For instance, the status bar might obscure a critical control in the application you are testing.
5. Click **Start Recording**.
   SilkTest performs the following actions:

   - Closes the **Record Testcase** dialog box.
   - Starts your application, if it was not already running.
   - Removes the editor window from the display.
   - Displays the **Recording status** window.
   - Waits for you to take further action.
6. Interact with your application, driving it to the state that you want to test.

   As you interact with your application, SilkTest records your interactions in the **Testcase Code** box of the **Record Testcase** dialog box, which is not visible.
7. To add a validation to the script, press **Ctrl+Alt**.

   Note: For any application that uses **Ctrl+Shift** as the shortcut key combination, press **Ctrl+Shift** to open the **Verify Properties** dialog box.

   The **Verify Properties** dialog box opens.
8. Check the check boxes for the properties that you want the script to verify and then click **Ok**.
   The recorder adds a verification action to the script for the locator to which you pointed when you pressed the shortcut key combination, **Ctrl+Alt** or **Ctrl+Shift**.
9. To review what you have recorded, click **Stop Recording** in the **Recording** window.
   SilkTest displays the **Record Testcase** dialog box, which contains the code that has been recorded for you.

10. To resume recording your interactions, click **Resume Recording** in the dialog box. To temporarily suspend recording, click **Stop Recording** on the **Recording** window.
11. Click **Paste to Editor**.
    If you have interacted with objects in your application that have not been identified in your include files, the **Update Files** dialog box opens.
12. Perform one of the following steps:

    - Click **Paste testcase and update window declaration(s)** and then click **OK**. In most cases, you want to choose this option.
    - Choose **Paste testcase only** and then click **OK**. This option does not update the window declarations in the INC file when it pastes the script to the Editor. If you previously recorded the window declarations related to this test case, choose this option.

Running a testcase

When you run a testcase, SilkTest interacts with the application by executing all the actions you specified in the testcase and testing whether all the features of the application performed as expected.

SilkTest always saves the suite, script, or testplan before running it if you made any changes to it since the last time you saved it. By default, SilkTest also saves all other open modified files whenever you run a script, suite, or testplan. To prevent this automatic saving of other open modified files, uncheck the **Save Files Before Running** check box in the **General Options** dialog.

1. Make sure that the testcase you want to run is in the active window.
2. Click **Run Testcase** on the **Basic Workflow** bar. If the workflow bar is not visible, choose **Workflows** > **Basic** to enable it.
3. SilkTest displays the **Run Testcase** dialog, which lists all the testcases contained in the current script.
4. Select a testcase and specify arguments, if necessary, in the **Arguments** field. Remember to separate multiple arguments with commas.
5. To wait one second after each interaction with the application under test is executed, check the **Animated Run Mode (Slow-Motion)** check box. Typically, you will only use this check box if you want to watch the testcase run. For instance, if you want to demonstrate a testcase to someone else, you might want to check this check box. Executions of the default base state and functions that include one of the following strings are not delayed:

    - Verify
    - Exists
    - Is
    - Get
    - Set
    - Print
    - ForceActiveXEnum
    - Wait
    - Sleep
6. To view results using the **Silk TrueLog Explorer**, check the **Enable TrueLog (for Classic Agent only)** check box. Click **TrueLog Options** to set the options you want to record.

    The **Silk TrueLog Explorer** works with the SilkTest Classic Agent only. Use the **Difference Viewer** to analyze results for testcases that use the SilkTest Open Agent.
7. Click **Run**. SilkTest runs the testcase and generates a results file.

    For the SilkTest Classic Agent, multiple tags are supported by Windows 2000 and Windows XP Agents. If you are running testcases using other Agents, you can run scripts that use declarations with multiple tags. To do this, check the **Disable Multiple Tag Feature** check box in the **Agent Options** dialog on the Compatibility tab. When you turn off multiple-tag support, 4Test discards all segments of a multiple tag except the first one.

Viewing test results

Whenever you run tests, SilkTest generates a results file, which indicates how many tests passed and how many failed, describes why tests failed, and provides summary information.

1. Click the **Explore Results** button on the **Basic Workflow** or the **Data Driven Workflow** bars.
2. On the **Results Files** dialog, navigate to the file name that you want to review and click **Open**.

By default, the results file has the same name as the executed script, suite, or testplan. To review a file in the **SilkTest TrueLog Explorer**, open a `.xlg` file. To review a SilkTest results file in SilkTest, open a `.res` file.

Classic Agent

This section includes the tasks that you can perform in the **Basic Workflow** toolbar, when you are using the Classic Agent.

Creating a New Project

You can create a new project and add the appropriate files to the project, or you can have SilkTest Classic automatically create a new project from an existing file.

Since each project is a unique testing environment, by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as an options set by opening SilkTest Classic and clicking **Options** > **Save New Options Set**. You can add the options set to your project.

To create a new project:

1. In SilkTest Classic, click **File** > **New Project**, or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click the type of project that you want to test.

   The type of project that you select determines the default Agent. For instance, if you specify that you want to create an Adobe Flex project, the Open Agent is automatically set as the default agent. SilkTest Classic uses the default agent when recording a test case, enabling extensions, or configuring an application.
3. Click **OK**.
4. On the **Create Project** dialog box, type the **Project Name** and **Description**.
5. Click **OK** to save your project in the default location, `<SilkTest Classic installation directory>\Project`.

   To save your project in a different location, click **Browse** and specify the folder in which you want to save your project.

   SilkTest Classic creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` files to this location and copies the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini` to a `projectname\extend` subdirectory. SilkTest Classic then creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources associated with this project.
6. Perform one of the following steps:

   - If your test uses the Open Agent, configure the application to set up the test environment.
   - If your test uses the Classic Agent, enable the appropriate extensions to test your application.

Enabling Extensions Automatically Using the Basic Workflow

An extension is a file that serves to extend the capabilities of, or data available to, a more basic program. SilkTest Classic provides extensions for testing applications that use non-standard controls in specific development and browser environments.

If you are testing a generic project that uses the Classic Agent, perform the following procedure to enable extensions:

1. Start the application or applet for which you want to enable extensions.
2. Start SilkTest Classic and make sure the basic workflow bar is visible. If it is not, click **Workflows** > **Basic** to enable it.

   If you do not see **Enable Extensions** on the workflow bar, ensure that the default agent is set to the Classic Agent.
3. If you are using SilkTest Classic projects, click **Project** and open your project or create a new project.
4. Click **Enable Extensions**.

   You cannot enable extensions for SilkTest Classic (`partner.exe`), the Classic Agent (`agent.exe`), or the Open Agent (`openAgent.exe`).
5. Select your test application from the list on the **Enable Extensions** dialog box, and then click **Select**.
6. If your test application does not display in the list, click **Refresh**. Or, you may need to add your application to this list in order to enable its extension.
7. Click **OK** on the **Extension Settings** dialog box, and then close and restart your application.
8. If you are testing an applet, the **Enable Applet Support** check box is checked by default.
9. When the **Test Extension Settings** dialog box opens, restart your application in the same way in which you opened it; for example, if you started your application by double-clicking the .exe, then restart it by double-clicking the .exe.
10. Make sure the application has finished loading, and then click **Test**.
    When the test is finished, a dialog box displays indicating that the extension has been successfully enabled and tested. You are now ready to begin testing your application or applet. If the test fails, review the troubleshooting topics.

When you enable extensions, SilkTest Classic adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

Setting the Recovery System for the Classic Agent

The recovery system ensures that each test case begins and ends with the application in its intended state. SilkTest Classic refers to this intended application state as the BaseState. The recovery system allows you to run tests unattended. When your application fails, the recovery system restores the application to the BaseState, so that the rest of your tests can continue to run unattended.

If you are testing an application that uses both the Classic Agent and the Open Agent, set the Agent that will start the application as the default Agent and then set the recovery system. If you use the Open Agent to start the application, set the recovery system for the Open Agent.

1. Make sure the application that you are testing is running.
2. Click **Set Recovery System** on the **Basic Workflow** bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
3. From the **Application** list, click the name of the application that you are testing.

   All open applications that are not minimized are listed. This list is dynamic and will update if you open a new application. If you are connected to the Open Agent, only those applications that have extensions enabled display in the list.

Note: If you selected a non-web application as the application:

- The **Command line** text box displays the path to the executable (`.exe`) for the application that you have selected.
- The **Working directory** text box displays the path of the application you selected.

If you selected a web application, the **Start testing on this page** text box displays the URL for the application you selected. If an application displays in the list, but the URL does not display in this text box, your extensions may not be enabled correctly. Click **Enable Extensions** in the **Basic Workflow** bar to automatically enable and test extension settings.

4. *Optional:* In the **Frame file name** text box, modify the frame file name and click **Browse** to specify the location in which you want to save this file.

   Frame files must have a .inc extension. By default, this field displays the default name and path of the frame file you are creating. The default is `frame.inc`. If `frame.inc` already exists, SilkTest appends the next logical number to the new frame file name; for example, `frame1.inc`.
5. *Optional:* In the **Window name** text box, change the window name to use a short name to identify your application.
6. Click **OK**.
7. Click **OK** when the message indicating that the recovery system is configured displays.
8. A new 4Test include file, `frame.inc`, opens in the **SilkTest Editor**. Click the plus sign in the file to see the contents of the frame file.
9. Record a test case.

Recording a Testcase With the Classic Agent

When you record a testcase with the Classic Agent, SilkTest uses hierarchical object recognition, a fast, easy method to create scripts. However, testcases that use dynamic object recognition are more robust and easy to maintain. You can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements. You can use both recognition methods within a single testcase if necessary.

1. Enable extensions and set up the recovery system.
2. Click **Record Testcase** on the Basic Workflow bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
3. Type the name of your testcase in the Testcase name field of the **Record Testcase** dialog. Testcase names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
4. Select **DefaultBaseState** in the **Application State** field to have the built-in recovery system restore the default BaseState before the testcase begins executing. If you chose **DefaultBaseState** as the application state, the testcase is recorded in the script file as: `testcase testcase_name ()`. If you chose another application state, the testcase is recorded as: `testcase testcase_name () appstate appstate_name`.
5. If you do not want SilkTest to display the status window it normally shows during playback when driving the application to the specified base state—perhaps because the status bar obscures a critical control in the application you are testing—uncheck the **Show AppState status window** check box.
6. Click **Start Recording**. SilkTest:

   - Closes the **Record Testcase** dialog.
   - Starts your application, if it was not already running.
   - Removes the editor window from the display.
   - Displays the **Record Status** window.
   - Waits for you to take further action

7. Interact with your application, driving it to the state that you want to test. As you interact with your application, SilkTest records your interactions in the **Testcase Code** field of the **Record Testcase** dialog, which is not visible.
8. To review what you have recorded, click the **Done** button in the **Record Status** window.

   SilkTest displays the **Record Testcase** dialog, which contains the 4Test code that has been recorded for you.
9. To resume recording your interactions, click the **Resume Recording** button in the dialog. To temporarily suspend recording, click the **Pause Recording** button on the **Record Status** window.
10. Verify the testcase.

Running a testcase

When you run a testcase, SilkTest interacts with the application by executing all the actions you specified in the testcase and testing whether all the features of the application performed as expected.

SilkTest always saves the suite, script, or testplan before running it if you made any changes to it since the last time you saved it. By default, SilkTest also saves all other open modified files whenever you run a script, suite, or testplan. To prevent this automatic saving of other open modified files, uncheck the **Save Files Before Running** check box in the **General Options** dialog.

1. Make sure that the testcase you want to run is in the active window.
2. Click **Run Testcase** on the **Basic Workflow** bar. If the workflow bar is not visible, choose **Workflows** > **Basic** to enable it.
3. SilkTest displays the **Run Testcase** dialog, which lists all the testcases contained in the current script.
4. Select a testcase and specify arguments, if necessary, in the **Arguments** field. Remember to separate multiple arguments with commas.
5. To wait one second after each interaction with the application under test is executed, check the **Animated Run Mode (Slow-Motion)** check box. Typically, you will only use this check box if you want to watch the testcase run. For instance, if you want to demonstrate a testcase to someone else, you might want to check this check box. Executions of the default base state and functions that include one of the following strings are not delayed:

   - Verify
   - Exists
   - Is
   - Get
   - Set
   - Print
   - ForceActiveXEnum
   - Wait
   - Sleep
6. To view results using the **Silk TrueLog Explorer**, check the **Enable TrueLog (for Classic Agent only)** check box. Click **TrueLog Options** to set the options you want to record.

   The **Silk TrueLog Explorer** works with the SilkTest Classic Agent only. Use the **Difference Viewer** to analyze results for testcases that use the SilkTest Open Agent.
7. Click **Run**. SilkTest runs the testcase and generates a results file.

   For the SilkTest Classic Agent, multiple tags are supported by Windows 2000 and Windows XP Agents. If you are running testcases using other Agents, you can run scripts that use declarations with multiple tags. To do this, check the **Disable Multiple Tag Feature** check box in the **Agent Options** dialog on the Compatibility tab. When you turn off multiple-tag support, 4Test discards all segments of a multiple tag except the first one.

Viewing test results

Whenever you run tests, SilkTest generates a results file, which indicates how many tests passed and how many failed, describes why tests failed, and provides summary information.

1. Click the **Explore Results** button on the **Basic Workflow** or the **Data Driven Workflow** bars.
2. On the **Results Files** dialog, navigate to the file name that you want to review and click **Open**.

By default, the results file has the same name as the executed script, suite, or testplan. To review a file in the **SilkTest TrueLog Explorer**, open a `.xlg` file. To review a SilkTest results file in SilkTest, open a `.res` file.

Accessing Sample Applications

SilkTest Classic provides several sample applications. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*.

After you have installed the sample applications, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** to select the sample application that you want to use.

SilkTest Projects

Overview of SilkTest Classic Projects

SilkTest Classic projects organize all the resources associated with a test set and present them visually in the **Project Explorer**, making it easy to see, manage, and work within your test environment.

SilkTest Classic projects store relevant information about your project, including the following:

- References to all the resources associated with a test set, such as plans, scripts, data, options sets, .ini files, results, frame files, and include files.
- Configuration information.
- Editor settings.
- Data files for attributes and queries.

All of this information is stored at the project level, meaning that once you add the appropriate files to your project and configure it once, you may never need to do it again. Switching among projects is easy - since you need to configure the project only once, you can simply open the project and run your tests.

When you create a new project, SilkTest Classic automatically uses the Agent that supports the type of application that you are testing. For instance, if you create an Adobe Flex project, SilkTest Classic uses the Open Agent.

Each project is a unique testing environment

By default, new projects do not contain any settings, such as enabled extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as a options set by opening SilkTest Classic and clicking **Options** > **Save New Options Set**. You can include the options set when you create your project. You can create a project manually or have SilkTest Classic automatically generate a project for you, based on existing files that you specify.

Storing project information

SilkTest Classic stores project-related information in the following two project files:

`projectname.vtp`	The project file has a Verify Test Project (.vtp) extension and is organized as an .ini file. It stores the names and locations of files used by the project.
`projectname.ini`	The project initialization file, similar to the `partner.ini` file, stores information about options sets, queries, and other resources included in your project.

These files are created in the `projectname` folder. When you create your project, SilkTest Classic prompts you to store your project in `<SilkTest Classic installation directory>\Project`, the default location. You can specify another location if you do not want to use the default project folder. SilkTest creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` files to this location, and copies the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini`, to the `projectname\extend` subdirectory.

If you are using Microsoft Windows Vista, project data is saved in the `C:\Users\<username>\AppData\Local\VirtualStore\Program Files\Silk\SilkTest <version>` folder, and not in the `Program Files` folder where SilkTest Classic is installed. When you open a project from SilkTest Classic, project data is displayed as expected. However, if you use the File Explorer to access the project, you need to click the **Compatibility Files** icon to access the project. To store project data in the installation folder, as with previous Windows versions, you can disable the Vista User Access Control.

When you export a project, the default location is the project directory.

Note: The extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini`, located in your `<SilkTest Classic installation directory>\extend` folder are copied to your `projectname\extend` directory, regardless of what extension you have enabled. Do not rename your `projectname\extend` directory; this directory must exist in order for SilkTest Classic to open your project.

You can have SilkTest Classic automatically enable the appropriate extension using the basic workflow bar, or you can manually enable extensions. The current project uses the extension options in the extension .ini file copied to the `projectname\extend` directory. Any modifications you make to the options for this enabled extension will be saved to the copy stored within the current project in your `projectname\extend` directory.

The `projectname\extend` directory is used only for local testing on the host machine. If you want to test on remote Agent machines, you must copy the .ini files from the `projectname\extend` directory to the `<SilkTest Classic installation directory>\extend` directory on the target machines.

File references

Whether you are emailing, packaging, or adding files to a project, it is important to understand how SilkTest Classic stores the path of the file. The .vtp files of SilkTest Classic use relative paths for files on the same root drive and absolute paths for files with different root drives. The use of relative and absolute file paths is not configurable and cannot be overridden. If you modify the .vtp file to change file references from relative paths to absolute paths, the next time you open and close the project it will have relative paths and your changes will be lost.

Accessing files within your project

Working with SilkTest Classic projects makes it easy to access your files - once you have added a file to your project, you can open it by double-clicking it in the **Project Explorer**. The **Project Explorer** contains the following two tabs:

Tab	Description
Files	Lists all of the files included in the project. From the **Files** tab, you can view, edit, add, and remove files from the project, as well as right-click to access menu options for each of the file types. From the **Files** tab, you can also add, rename, remove and work with folders within each category.
Global	Displays all the resources that are defined at a global level within the project's files. For example test cases, functions, classes, window declarations, and others. When you double-click an object on the **Global** tab, the file in which the object is defined opens and your cursor displays at the beginning of the line in which the object is defined. You can run and debug test cases and application states from the **Global** tab. You can also sort the elements that display within the folders on the **Global** tab.

Existing test sets do not display in the **Project Explorer** by default; you must convert them into projects.

Sharing a project among a group

Apply the following guidelines when sharing a project among a group:

- Create the project in the location from which it will be shared; for example on a network drive.
- Ensure that testers create the same directory structure on their machines.

Project Explorer

Use the **Project Explorer** to view and work with all the resources within a SilkTest Classic project. You can access the **Project Explorer** by clicking:

- **File** > **Open Project** and specifying the project you want to open.
- **File** > **New Project** and creating a new project.
- **Project** > **View Explorer**, if you currently have a project open and do not have the **Project Explorer** view on.
- **Project** > **New Project** or **Open Project** on the **Basic Workflow** bar.

The resources associated with the project are grouped into categories. You can easily navigate among and access all of these resources using the **Files** and **Global** tabs. When you double-click a file on the **Files** tab, or an object on the **Global** tab, the file opens in the right pane. You can drag the divider to adjust the size of the **Project Explorer** windows and click **Project** > **Align** to change the orientation of the tabs from left to right.

Files tab

The **Files** tab lists all of the files that have been added to the project. The file name displays first, followed by the path. If files exist on a network drive, they are referenced using Universal Naming Conventions (UNC). Files are grouped into the following categories:

Category	Description
Profile	Contains project-specific initialization files, such as the `projectname.ini` and option sets files, which means .opt files, that are associated with the project.
Script	Contains test scripts, which means .t and .g.t files, that are associated with the project.
Include/Frame	Contains include files, which means .inc files, and frame/object files that are associated with the project.
Plan	Contains test plans and suite files, which means .pln and .s files, that are associated with the project.
Results	Contains results, which means .res and .rex files, that are associated with the project.
Data	Contains data associated with the project, such as Microsoft Word documents, text files, bitmaps, and others. Double-click the file to open it in the appropriate application. You must open files that are not associated with application types in the Windows Registry using the **File/Open** dialog box.

From the **Files** tab, you can view, edit, add, remove and work with files within the project. For example, to add a file to the project, right-click the category name, for example Script, and then click **Add File**. After you have added the file, you can right-click the file name to view options for working with the file, such as record test case and run test case. SilkTest Classic functionality has not changed - it is now accessible through a project.

You can work with the folders within the categories on the **Files** tab, by adding, renaming, moving, and deleting folders within each category.

Global tab

The **Global** tab lists resources that are defined at a global level within the entire project. The resource name displays first, followed by the file in which it is defined. Resources contained within the project's files are grouped into the following categories:

- Records
- Classes

- Enums
- Window Declarations
- Testcases
- Appstates
- Functions
- Constants

From the **Global** tab, you can go directly to the location in which a global object or resource is defined. Double-click any object within the folders to go to the location in which the object is defined. SilkTest Classic opens the file and positions your cursor at the beginning of the line in which the object is defined.

You can also run and debug test cases and application states by right-clicking a test case or application state, and then selecting the appropriate option. For example, right-click a test case within the `Testcase` folder and then click **Run**. SilkTest Classic opens the file containing the test case you selected, and displays the **Run Testcase** dialog box with the selected test case highlighted. You can input argument values and run or debug the test case.

On the **Global** tab, you can sort the resources within each node by resource name, file name, or file date.

Note: Methods and properties are not listed on the **Global** tab since they are specific to classes or window declarations. You can access methods and properties by double-clicking the class or window declaration in which they are defined.

You cannot move files within the **Project Explorer**. For example, you cannot drag a script file under the **Frame** file node. However, you can drag the file to another folder within the same category node.

Note: If you change the location or name of a file included in your project, outside of SilkTest Classic, you must make sure the `projectname.vtp` contains the correct reference.

Creating a New Project

You can create a new project and add the appropriate files to the project, or you can have SilkTest Classic automatically create a new project from an existing file.

Since each project is a unique testing environment, by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as an options set by opening SilkTest Classic and clicking **Options** > **Save New Options Set**. You can add the options set to your project.

To create a new project:

1. In SilkTest Classic, click **File** > **New Project**, or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click the type of project that you want to test.

   The type of project that you select determines the default Agent. For instance, if you specify that you want to create an Adobe Flex project, the Open Agent is automatically set as the default agent. SilkTest Classic uses the default agent when recording a test case, enabling extensions, or configuring an application.
3. Click **OK**.
4. On the **Create Project** dialog box, type the **Project Name** and **Description**.
5. Click **OK** to save your project in the default location, `<SilkTest Classic installation directory>\Project`.

   To save your project in a different location, click **Browse** and specify the folder in which you want to save your project.

   SilkTest Classic creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` files to this location and copies the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini` to a `projectname\extend` subdirectory. SilkTest

Classic then creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources associated with this project.

6. Perform one of the following steps:
   - If your test uses the Open Agent, configure the application to set up the test environment.
   - If your test uses the Classic Agent, enable the appropriate extensions to test your application.

Overview of AutoGenerate Project

SilkTest Classic can automatically generate a new project from an existing file or files, such as plan, script, or suite files. SilkTest Classic scans the specified files, gathers all references to other files contained within them, for example references to sub-plans, include files, options sets, scripts, window children, or functions, and adds them to the project that it generates. SilkTest Classic searches for the following statements within files:

- use
- include:
- script:
- framefile:
- optionset:
- usefiles=

If SilkTest Classic cannot find a file that is referenced in the files from which you are automatically generating your project, an error message displays noting the missing file name and the file and line number containing the reference to the missing file.

Result files, data files, bitmaps, and files included or referenced through `File` functions, are not automatically added to your project. You must add these files manually.

AutoGenerate and options sets

If an .opt file is referenced in a .pln file or selected as a file to generate from, AutoGenerate will parse the .opt file, find references to UseFiles=, and include any files referenced here in the **Project Explorer**, with the exception of .inc located in the SilkTest Classic extend directory. It will not automatically add …`\extend\filename.inc` files to the project. However, you can manually add these files to your project.

If you add an options set to the project and want to use it, you must load it into memory. On the **Files** tab, expand the **Profile** node, right-click the options set you want to load, and then click **Open Options Set**.

Note: Double-clicking an options set opens it for editing, but does not load it into memory.

Specifying project settings

Remember that each project is a unique testing environment, so by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. You can specify your settings once in your new project using the **Options** dialog boxes, or you can retain the settings from your current test set, by saving them as an options set. Open SilkTest Classic and click **Options** > **Save New Options Set**. You can include the options set when you create your project.

Note: The `testplan.ini` files are not included in options sets so you must specify your `testplan.ini` file manually.

Automatically Generating a New Project

SilkTest Classic can automatically generate a new project from an existing file or files, such as plan, script, or suite files. For additional information, see *Overview of AutoGenerate Project*.

To have SilkTest Classic automatically generate a new project from the files you specify:

1. Click **File** > **New Project** or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click **Autogenerate**.
3. On the **AutoGenerate Project** dialog box, type the **Project Name** and **Description**.
4. Specify the location where you want to save your project.
   We recommend the default location, `<SilkTest Classic installation directory>\Project`. SilkTest Classic will create a `projectname` folder within this directory, save the `projectname.vtp` and `projectname.ini` files to this location and copy the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini`, to a `projectname\extend` subdirectory. If you do not want to save your project in the default location, click **Browse** and specify the folder in which you want to save your project.
5. Click **Add** to select the file from which you want SilkTest Classic to generate a project, for example, a master plan file, and then click **Open**.
   The file is added to the list in the **Generate from** list box.
6. *Optional:* Click **Add** again to select additional files, if applicable.
7. *Optional:* To add an options set, click **Add**, select **All Files** in the **Files of type** list box, select the options set you want to add, and then click **Open**.
8. *Optional:* To remove a file from the list, select the file, and then click **Remove**.
9. Click **OK** when you have finished adding files.

SilkTest Classic automatically generates a new project from the file or files you selected. This may take a few moments, depending on the size and number of files you are converting into projects. SilkTest Classic creates a project, `projectname.vtp` and `projectname.ini`, and automatically includes all the files and resources that are referenced within the selected files in the project. You can view the files and resources that are automatically included in the project on the **Files** and **Global** tabs of the **Project Explorer**.

Opening an Existing Project

You can open a SilkTest Classic project as well as open an archived SilkTest Classic project. You can also open a SilkTest Classic project or archived project through the command line.

To open an existing project:

1. Click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar.
   If you already have a project open, a dialog box opens informing you that the open project will be closed. If you associated SilkTest Classic file types with SilkTest Classic during installation, then you can open a SilkTest Classic project or package by double-clicking the .vtp or .stp file.
2. If you are opening a packaged SilkTest Classic project, which means an .stp file, you must perform the following steps:
   a) Indicate into what directory you want to unpack the project in the **Base path** text box.
      The files are unpacked to the directory you indicate in the **Base path** text box.
   b) Enter a password into the **Password** text box if the archived SilkTest Classic project was saved with a password.

      If you open a package by double-clicking the .stp file, the base path is the directory that contains the .stp file.

      When you select a location for unpacking the archive on the **Open Project** dialog box, SilkTest Classic uses that directory path, the base path, to substitute for the drive and root directory in the Use Path and Use Files paths.

      The **Base path** and **Password** text boxes are enabled only if you are opening an .stp file.
3. On the **Open Project** dialog box, specify the project that you want to open, and then click **Open**.

If you open a project file (.vtp) by clicking **File** > **Open** command, the `projectname.vtp` file will open in the **4Test Editor**, but the project and its associated settings will not be loaded. Projects do not display in the recently opened files list. To close all open files within a project, click **Window** > **Close All**.

Converting Existing Tests to a Project

Since each project is a unique testing environment, by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as an options set by clicking **Options** > **Save New Options Set**. You can include the options set when you create your project.

To convert existing test sets to a project:

1. Choose **File** > **New Project** or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click **AutoGenerate**.
3. On the **AutoGenerate Project** dialog box, type the **Project Name** and **Description**.
4. Specify the location where you want to save your project.

   We recommend the default location, `<SilkTest Classic installation directory>\Project`.

   SilkTest Classic will create a `projectname` folder within this directory, save the `projectname.vtp` and `projectname.ini` to this location and copy the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini`, to a `projectname\extend` subdirectory. If you do not want to save your project in the default location, click **Browse** and specify the folder in which you want to save your project.
5. Click **Add** to select the file from which you want SilkTest Classic to generate a project, for example, a master plan file, and then click **Open**.
   The file is added to the list in the **Generate from** list box.
6. *Optional:* Click **Add** again to select additional files, if applicable.
7. *Optional:* To add an options set, click **Add**, select **All Files** in the **Files of type** list box, select the options set you want to add, and then click **Open**.
8. *Optional:* To remove a file from the list, select the file, and then click **Remove**.
9. Click **OK** when you have finished adding files.
   SilkTest Classic automatically generates a new project from the file or files you selected. This may take a few moments, depending on the size and number of files you are converting into projects. SilkTest Classic creates the files `projectname.vtp` and `projectname.ini` in the location that you have you specified in the **Save in** text box and displays the files and resources included in the automatically-generated project in the **Project Explorer**.
10. If you have added an options set and want to use it, you must load it into memory: on the **Files** tab, expand the **Profile** node, right-click the options set you want to load, and then click **Open Options Set**.

    Note: Double-clicking an options set opens it for editing, but does not load it into memory.

Results files, data files, bitmaps, and files included or referenced through file functions are not automatically added to your project. You must add these files manually. You can also convert existing tests to a project by creating a new project and manually adding the files to the project.

Using Option Sets in Your Project

To use an options set within your project, you must make sure that the options set is loaded into memory. You can tell if an options set is loaded by looking at the SilkTest Classic title bar. If `filename.opt`

displays in the title bar, then the options set `filename.opt` is loaded. If an options set is loaded, it overrides the settings contained in the `projectname.ini` file.

Note: When an options set is loaded, the context menu options are available only for the loaded options set; these menu options are grayed out for .ini and .opt files that are not loaded.

You can load an options set into your project using any of the following methods:

- If the options set is included in your project, within the **Profile** node on the **Files** tab, right-click the options set that you want to load and then click **Open Options Set**.
- Right-click **Save New Options Set** to load the options set and add it under the **Profile** node on the **Files** tab.
- Use the **Options** menu; click **Options** > **Open Options Set**, browse to the options set (.opt) that you want to load, and then click **Open**.
- Load the options set at runtime using the optionset keyword. This loads the options set at the point in the plan file in which the options set is called. All test cases that follow use this options set.

If an options set was loaded when you closed SilkTest Classic, SilkTest Classic automatically re-loads this options set when you re-start SilkTest Classic.

To include an options set in your project, you can add the options set by right-clicking **Profile** on the **Files** tab, clicking **Add File**, selecting the options set you want to add to the project, and then clicking **OK**. You can also click **Save New Options Set**; this loads the options set and adds it under the **Profile** node on the **Files** tab.

Editing an Options Set

To edit an options set in your project:

1. On the **Files** tab, expand the **Profile** node.
2. Right-click the options set that you want to edit and click **Open Options Set**.
   The options set is loaded into memory.
3. Right-click the options set that you want to edit again and select the type of option you want to edit.
   For example Runtime, Agent, Extensions, and others.
4. Modify your options and then click **OK**.
   Your current settings are changed and saved to the .opt file.

If you want to change settings for future use, double-click the options set that you want to edit on the **Files** tab. This opens the options file in the Editor without loading the options file into memory. Changes you make to the options set in the Editor will be saved, but will not take effect until you load the options set by selecting **Open Options Set** from the **Options** menu or the right-click shortcut.

SilkTest Classic File Types

SilkTest Classic uses the following types of files in the automated testing process, each with a specific function. The files marked with an * are required by SilkTest Classic to create and run test cases.

File Type	Extension	Description
Project	.vtp	SilkTest Classic projects organize all the resources associated with a test set and present them visually in the **Project Explorer**, making it easy to see, manage, and work within your test environment.

File Type	Extension	Description
		The project file has a Verify Test Project (`.vtp`) extension and is organized as an `.ini` file; it stores the names and locations of files used by the project. Each project file also has an associated project initialization file: `projectname.ini`.
Testplan	.pln	An automated test plan is an outline that organizes and enhances the testing process, references test cases, and allows execution of test cases according to the test plan detail. It can be of type masterplan or of subplan that is referenced by a masterplan.
Test Frame*	.inc	A specific kind of include file that upon creation automatically captures a declaration of the AUT's main window including the URL of the Web application or path and executable name for client/server applications; acts as a central repository of information about the AUT; can also include declarations for other windows, as well as application states, variables, and constants.
4Test Script*	.t	Contains recorded and hand-written automated test cases, written in 4Test language, that verify the behavior of the AUT.
Data driven Script	.g.t	Contains data driven test cases that pull their data from databases.
4Test Include File	.inc	A file that contains window declarations, constants, variables, classes, and user defined functions.
Suite	.s	Allows sequential execution of several test scripts.
Text File	.txt	An ASCII file that can be used for the following: • Store data that will be used to drive a data driven test case. • Print a file in another document (Word) or presentation (PowerPoint). • Accompany your automation as a readme file. • Transform a tab-delimited plan into a SilkTest Classic plan.
Results File	.res	Is automatically created to store a history of results for a test plan or script execution.

Organizing

This section includes the topics that are available for organizing SilkTest Classic projects.

Adding Existing Files to a Project

You can add existing files to a project or create new files to add to the project. We recommend adding all referenced files to your project so that you can easily see and access the files, and the objects contained within them. Referenced files do not have to be included in the project. Plans and scripts will continue to run, provided the paths that are referenced are accurate.

When you add a file to a project, project files (.vtp files) use relative paths for files on the same root drive and absolute paths for files with different root drives. The use of relative and absolute files is not configurable and cannot be overridden.

To add an existing file to a project:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar, select the project to which you want to add a file, and then click **Open**.
2. On the **Project Explorer**, select the **Files** tab, right-click the node associated with the type of file you want to add, and then click **Add File**.

   For example, to add a script file to the project, right-click **Script**, and then click **Add File**.
3. On the **Add File to Project** dialog box, specify the file you want to add to the open project, and then click **Open**.

The file name, followed by the path, displays under the appropriate category on the **Files** tab sorted alphabetically by name and is associated with the project through the `projectname.vtp` file. If files exist on a network drive, they are referenced using Universal Naming Conventions (UNC).

You can also add existing files to the project by clicking **Project** > **Add File**. SilkTest Classic automatically places the file in the appropriate node, based on the file type; for example if you add a file with a .pln extension, it will display under the **Plan** node on the **Files** tab. We do not recommend adding application .ini files or SilkTest Classic .ini files, which are `qaplans.ini`, `propset.ini`, and the `extension.ini` files, to your project. If you add object files, which are .to and .ino files, to your project, the files will display under the **Data** node on the **Files** tab. Objects defined in object files will not display in the **Global** tab. You cannot modify object files within the SilkTest Classic editor because object files are binary. To modify an object file, open the source file, which is a .t or .inc file, edit it, and then recompile.

Renaming Your Project

The `projectname.ini` and the `projectname.vtp` refer to each other; make sure the references are correct in both files when you rename your project.

To rename your project:

1. Make sure the project you want to rename is closed.
2. In Windows Explorer, locate the `projectname.vtp` and `projectname.ini` associated with the project name you want to change.
3. Change the names of `projectname.vtp` and `projectname.ini`. Make sure that you use the same projectname for both files.
4. In a text editor outside of SilkTest Classic, open `projectname.vtp`, change the reference to the `projectname.ini` file to the new name, and then save and close the file. Do not open the project in SilkTest Classic yet.
5. In a text editor outside of SilkTest Classic, open `projectname.ini`, change the reference to the `projectname.vtp` file to the new name, and then save and close the file.
6. In SilkTest Classic, open the project by clicking **File** > **Open Project** or **Open ProjectOpen Project** on the basic workflow bar.
   The new project name displays.

Working with Folders in a Project

In addition to working with files, you can also add your own folders to all nodes listed on the **File** tab of the **Project Explorer**. For example, the **Files** tab of the **Project Explorer** can include notes.

You can also right-click a folder and click the following:

- **Expand All** to display all contents of a folder.
- **Collapse All** to collapse the contents of the folder.
- **Display Full Path** to show the full path for the contents.
- **Display Date/Time** to show creation information for the content file.

Adding a Folder to the Files Tab of the Project Explorer

You may add a folder to any of the categories (nodes) on the **Files** tab of the **Project Explorer**. You may not add a folder to the root project folder, nor change the titles of the root nodes.

To add a folder to a project:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. In the **Project Explorer**, click the **Files** tab, right-click a folder and select **Add Folder**.

3. On the **Add Folder** dialog box, enter the name of the new folder, then click **OK**.

   When you are naming a folder, you may use alphanumeric characters, underscore character, character space, or hyphens. Folder names may be a maximum of 256 characters long. Creating folders with more than 256 characters is possible, but SilkTest Classic will truncate the name when you save the project. The concatenated length of the names of all folders within a project may not exceed 256 characters. You may not use periods or parentheses in folder names. Within a node, folder names must be unique.

Moving Files and Folders

You may move an individual file or files between folders within the same category on the **Files** tab of the **Project Explorer**. You cannot move the predefined SilkTest Classic folders (nodes) such as Profile Script, Plan, Frame, and Data.

You may also move sub-folders within the same category on the **Files** tab. You cannot move sub-folders across categories.

To move a folder or file:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. In the **Project Explorer**, click the **Files** tab. Click a file, a folder, or shift-click to select several files or folders, then drag the items to the new location.
3. Release the mouse to move the items.

   There is no undo.

Removing a Folder from the Files tab of the Project Explorer

You may delete folders on the **Files** tab of the **Project Explorer**, however, you may not delete any of the predefined SilkTest Classic categories (nodes) such as Profile Script, Plan, Frame, and Data.

Note: There is no undo.

To remove a folder:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. In the **Project Explorer**, click the **Files** tab, right-click a folder and select **Remove Folder** to delete it from the **Project Explorer**.
   If you select a folder with child folders or a folder that contains items, SilkTest Classic displays a warning before deleting the folder.

Renaming a Folder on the Files Tab of the Project Explorer

You may rename any folder that you have added to a project. You may not rename any of the predefined SilkTest Classic folders (nodes) such as Profile, Script, Include/Frame, Plan, Results, or Data.

To rename a folder:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. In the **Project Explorer**, click the **Files** tab, then navigate to the folder you want to rename.
3. Right-click the folder and select **Rename Folder**.
4. On the **Rename Folder** dialog box, enter the new name of the folder then click **OK**.

   When naming a folder, you may use alphanumeric characters, underscore character, character space, or hyphens. Folder names may be a maximum of 64 characters long. You may not use periods or parentheses in folder names. Within a node, folder names must be unique.

Sorting Resources within the Global Tab of the Project Explorer

On the **Global** tab of the **Project Explorer**, you can sort the resources within each category (node) by resource name, file name, or file date.

To sort resources:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar, select the project whose elements you want to sort, and then click **Open**.
2. On the **Project Explorer**, click the **Global** tab, right-click the node associated with the type of element you want to sort, and then click **Sort by FileName** or **Sort by FileDate**.

   The default is sort by element name.
3. Click **Ascending** or **Descending** to indicate how you want to organize the sort.

   For example, to sort the elements of a script file by file date in reverse chronological order, right-click the **Script** node and select **Sort by FileDate**, then click **Descending**.

   When you release the mouse, the elements are sorted by the parameters you selected.

Moving Files Between Projects

We recommend that you use **Export Project** to move projects, but if you want to move only a few files rather than an entire project, you can open the project in SilkTest Classic and remove the files that you want to move from the project. Move the files to their new location in Windows Explorer, and then add the files back to the currently open project.

You can also move your project by opening the `projectname.vtp` and `projectname.ini` files in a text editor outside of SilkTest Classic and updating references to the location of source files. However, we recommend that you have strong knowledge of your files and how the partner and projectname .ini files work before attempting this. We advise you to use great caution if you decide to edit the projectname .vtp and projectname .ini files.

Removing Files from a Project

You cannot remove the `projectname.ini` file.

To remove a file from a project:

1. Click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar.
2. Click the plus sign [+] to expand the node associated with the type of file you want to remove, and then choose one of the following:
   - Right-click the file you want to remove, and then click **Remove File**.
   - Select the file in the **Project Explorer** and press the **Delete** key.
   - Select the file you want to remove on the **Files** tab, and then click **Project** > **Remove File**.

The file is removed from the project and references to the file are deleted from the `projectname.vtp`. The file itself is not deleted; it is just removed from the project.

Turning the Project Explorer View On and Off

The **Project Explorer** view is the default. If you do not want to view the **Project Explorer**, uncheck **Project** > **View Explorer**. You can continue to work with your files within the project, you just will not see the **Project Explorer**.

To turn **Project Explorer** view on, check **Project** > **View Explorer**.

If you do not want to use projects in SilkTest Classic, close the open project, if any, by clicking **File** > **Close Project**, and then use SilkTest Classic as you would have in the past.

Viewing Resources Within a Project

1. Click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar and select the project that you want to open.
2. Click one of the following:
   - The **Files** tab to view all the files associated with the open project.
   - The **Global** tab to view global objects defined in the files associated with the project.
3. To close all open files within a project, click **Window** > **Close All**.

Packaging a SilkTest Classic Project

You can package your SilkTest Classic project into a single compressed file that you can relocate to a different computer. When you unpack your project you will have a fully functional set of test files. This is useful if you need to relocate a project, email a project to a co-worker, or send a project to technical support.

Source files included in the packaged project

When you package a project, SilkTest Classic includes all of the source files, meaning the related files used by a project, such as:

Description	Extension
plan files	.pln
script files	.t
include files	.inc
suite files	.s
results files (optional)	.res and .rex
data files	-

SilkTest Classic takes these files and bundles them up into a new file with an .stp extension. The .stp file includes the configuration files necessary for SilkTest Classic to set up the proper testing environment such as `project.ini`, `testplan.ini`, optionset .opt files, and any .ini files found in the …`\project\<projectname>\extend` directory.

You have the option of including .res and .rex files when you package a SilkTest Classic project because these files are sometimes quite large and not necessary to run the project.

Relative paths in comparison to absolute paths

When you work with SilkTest Classic projects, the files that make up the project are identified by pathnames that are either absolute or relative. A relative pathname begins at a current folder or some number of folders up the hierarchy and specifies the file's location from there. An absolute pathname begins at the root of the file system (the topmost folder) and fully specifies the file's location from there. For example:

Absolute path `c:\Program Files\<SilkTest Classic install directory>\SilkTest\options.ini`

Relative path `..\tesla\SilkTest\options\options.ini` or `SUSDir\options.inc`

When you package a project, SilkTest Classic checks to make sure that the paths used within the project are properly maintained. If you try to compress a project containing ambiguous paths, SilkTest Classic displays a warning message. SilkTest Classic tracks the paths in a project in a log file.

Including all files needed to run tests

Files associated with a project, but not necessary to run tests, for example bitmap or document files, which you have manually added to the project are included when SilkTest Classic packages a project.

If SilkTest Classic finds any include:, script:, or use: statements in the project files that refer to files with absolute paths, `c:\program files\Silk\SilkTest\`, SilkTest Classic verifies if you have checked the **Use links for absolute files?** check box on the **Export Project** or on the **Email Project** dialog boxes.

- If you check the **Use links for absolute files?** check box, SilkTest Classic treats any file referenced by an absolute path in an include, script, or use statement as a placeholder and does not include those files in the package. For example, if there are use files within the **Runtime Options** dialog box referred to as "q:\qaplans\SilkTest\frame.inc" or "c", these files are not included in the package. The assumption is that these files will also be available from wherever you unpack the project.
- If you uncheck the **Use links for absolute files?** check box, SilkTest Classic includes the files referenced by absolute paths in the packaged project. For example, if the original file is stored on `c:\temp\myfile.t`, when unpacked at the new location, the file is placed on `c:\temp\myfile.t`.

The following table compares the results of packaging projects based on whether there are any absolute file references in your source files, and how you respond to the **Use links for absolute files?** check box on the **Export Project** or on the **Email Project** dialog boxes.

Any absolute references in source files?	Use links for absolute files?	Results
No	Checked or unchecked	Package unpacks to any location.
Yes	Checked	Files referenced by absolute paths are not included in the packaged project.
Yes	Unchecked	Files referenced by absolute paths are put into a ZIP file within the packaged project.

Note:

- If there are any source files located on a different drive than the .vtp project file, and if there are files referenced by absolute paths in the source files, SilkTest Classic treats the source files as referenced by absolute paths. The assumption is that the absolute paths will be available from the new location. SilkTest Classic therefore puts the files into a zip file within the packaged project for you to unpack after you unpack the project.
- Files not included in the package - The assumption is that since these files are referenced by absolute paths, these same files and paths will be available when the files are unpacked. On unpacking, SilkTest Classic warns you about these files and lists them in a log file (manifest.XXX).
- ip files – Because you elected not to use links for files referenced by absolute paths, these files are put into a zip file within the packaged project. The zip file is named with the root of the absolute path. For example, if the files are located on c:/, the zip file is named `c.zip`.

Tips for successful packaging and unpacking

For best results when packaging and unpacking SilkTest Classic projects:

- Put your .vtp project file and source files on the same drive.
- Use relative paths to reference the following:
- • include statements

    - options sets
    - use paths set within the **Runtime Options** dialog box
    - use statements in 4Test scripts
    - script statements
- Uncheck the default **Use links for absolute files?** check box if your source files are on a different drive as the .vtp project file and if there are files referenced by absolute paths in your source files.

Packaging with SilkTest Classic Runtime and the Agent

If you are running SilkTest Classic Runtime, you may not package or email a project.

If you are running the Agent, you may package or email a project.

Emailing a Packaged Project

Emailing a project automatically packages a SilkTest Classic project and then emails it to an email address. In order for this to work, you must have an email client installed on the computer that is running SilkTest Classic.

You cannot email a project if you are running SilkTest Classic Runtime.

One of the options you can select before emailing is to compile your project. If a compile error occurs, SilkTest Classic displays a warning message, and you can opt to continue or to cancel the email.

SilkTest Classic supports any MAPI-compliant e-mail clients such as:

- Outlook Express
- Netscape Messenger
- Eudora

The maximum size for the emailed project is determined by your e-mail client. SilkTest Classic does not place any limits on the size of the project.

To email your project:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. Click **File** > **Email Project**.

   You can only email a project if you have that project open.
3. On the **Email Project** dialog box, type the email address where you want to send the SilkTest Classic project.

   For example, enter `support@acme.com` to send a package to Acme Technical Support. You do not have to specify an email address here; your email program will prompt you for one before sending the email.
4. Select the options for the package you want to email.

   For an explanation of these options, see the description of the **Email Project** dialog box. The **Email Address** text box is required, though you can edit it later.
5. Click **OK**.
   If you opted to compile the project before packaging it, SilkTest Classic displays a warning message if any file failed to compile. SilkTest Classic opens a new email message and attaches the packaged project to a message. You can edit the recipient, add a subject line, and text, just as you can for any outgoing message.
6. Click **Send** to add the project to your outgoing queue.
   If your email client is already open, your message is sent automatically. If your email client was not open, the message is placed in your outgoing queue.

Note: If you have a crash during the email process, we recommend deleting the partially packaged project or draft email message, if any, and starting the process again.

Exporting a Project

Exporting a SilkTest Classic project lets you copy all the files associated with a project to a directory or a single compressed file in a directory.

You cannot export a project if you are running SilkTest Classic Runtime.

SilkTest Classic will not change the file creation dates when copying the project's files.

One of the options you can select before exporting is to compile your project. If a compile error occurs, SilkTest Classic displays a warning message, and you can opt to continue or to cancel the compile.

To export your project:

1. If your project is not already open, click **File** > **Open Project** or click **Open Project** > **Open Project** on the basic workflow bar. Select a project, then click **Open**.
2. Click **File** > **Export Project**.
   You can only export a project if you have the project open.
3. On the **Export Project** dialog box, enter the directory to which you want to export the project or click  to locate the export folder.
   The default location is the parent directory of the project folder, which means the folder containing the project file, not the project's current location.
4. Check the **Export to single SilkTest Classic package** check box if you want to package the SilkTest Classic project into a single compressed file.
5. In the **Options** area, select the appropriate options for your project.
   For an explanation of these options, see the description of the **Export Project** dialog box.

   Note: Using references for absolute paths produces a smaller package that can be opened more quickly.
6. Click **OK**.
   SilkTest Classic determines all the files necessary for the project and copies them to the selected directory or compresses them into a package. SilkTest Classic displays a warning message if any of the files could not be successfully packaged and gives you the option of continuing.

If you have a crash during the export process, we recommend deleting the partially packaged project, if any, and starting the process over again.

Troubleshooting

SilkTest Classic cannot find files when opening my project

If, when opening your project, SilkTest Classic cannot find a file in the location referenced in the project file (.vtp), an error message displays noting the file that cannot be found.

SilkTest Classic may not be able to find files that have been moved or renamed outside of SilkTest Classic, for example in Windows Explorer, or files that are located on a shared network folder that is no longer accessible.

- If SilkTest Classic cannot find a file in your project, we suggest that you note the name of missing file, and click **OK**. SilkTest Classic will open the project and remove the file that it cannot find from the project list. You can then add the missing file to your project.
- If SilkTest Classic cannot open multiple files in your project, we suggest you click **Cancel** and determine why the files cannot be found; for example a directory was moved. Depending upon the

problem, you can determine how to make the files accessible to the project. You may need to add the files from their new location, or, if all the source files have been moved, autogenerate a new project.

SilkTest Classic cannot find a file when automatically generating a new project

If SilkTest Classic cannot find a file that is referenced in the files from which you are automatically generating your project, an error message displays noting the missing file name and the file and line number containing the reference to the missing file.

Note the name of the missing file, and then click **OK**. You can then locate and add the missing file or remove the reference to it from your files.

SilkTest Classic did not add my .inc file when auto-generating my project

If an .opt file is referenced in a .pln file or selected as a file to generate from, AutoGenerate will parse the .opt file, find references to `UseFiles=`, and include any files referenced here in the Project Explorer, with the exception of .inc located in the SilkTest Classic extend directory. It will not automatically add … `\extend\filename.inc` files to the project. However, you can manually add these files to your project.

AutoGenerate only parses .opt files looking for `UseFiles=`, not .ini files.

SilkTest Classic cannot load my project file; the contents of my projectname.vtp have changed or my projectname.ini file has been moved

If you remove or incorrectly edit the `ProjectIni=` line in the `ProjectProfile` section of your `projectname.vtp` file, or if you have moved your `projectname.ini` file and this line no longer points to the correct location of your `projectname.ini` file, SilkTest Classic will not be able to load your project.

Make sure that `ProjectProfile` section exists in your `projectname.vtp` file and refers to the correct name and location of your `projectname.ini` file. The `projectname.ini` and the `projectname.vtp` refer to each other; make sure the references are correct in both files. Make any necessary changes in a text editor outside of SilkTest Classic.

Here is a sample `ProjectProfile` section in a `projectname.vtp` file:

```
[ProjectProfile]
ProjectIni=C:\Program Files\<SilkTest install directory>\SilkTest\Projects
\ProjectName.ini
```

SilkTest Classic cannot save files to my project

You cannot add or remove files from a read-only project. If you attempt to make any changes to a read-only project, a message box displays indicating that your changes will not be saved to the project.

For example, `Unable to save changes to the current project. The project file has read-only attributes.`

When you click **OK** on the error message box, SilkTest adds or removes the file from the project temporarily for that session, but when you close the project, the message box displays again. When you re-open the project, you will see your files have not been added or removed.

SilkTest Classic does not run

If SilkTest does not run because it is looking for the following:	You can do the following:
Project files that may have been moved or corrupted.	Open the `partner.ini` file in a text editor and remove the `CurrentProject=` line from the `ProjectState` section. SilkTest should then open, however your project will not open. You can examine

If SilkTest does not run because it is looking for the following:	You can do the following:

	your `projectname.ini` and the `projectname.vtp` files to determine and correct the problem.
	Here is a sample `ProjectProfile` section:
	`[ProjectState]`
	`CurrentProject=C:\Program Files \<SilkTest install directory>`
	`\SilkTest\Examples\ProjectName.vtp`

A `testplan.ini` file that has been corrupted.	Delete or rename the corrupted `testplan.ini` file, and then restart SilkTest.

My files no longer display in the Recently Opened Files list box

After you open or create a project, files that you had recently opened outside of the project do not display in the **Recently Opened Files** list box.

Cannot find items in Classic 4Test

If you are working with Classic 4Test, objects display in the correct nodes on the **Global** tab, however when you double-click an object, the file opens and the cursor displays at the top of the file, instead of in the line in which the object is defined.

Editing the projectname.vtp and projectname.ini files

We recommend that you have strong knowledge of your files and how the partner and `projectname.ini` files work before attempting to edit these files. We advise you to use great caution when editing the `projectname.vtp` and `projectname.ini` files.

1. Make sure that your project is closed and that all the files referenced by the project exist.
2. Open the `projectname.vtp` and `projectname.ini` files in a text editor outside of SilkTest Classic. Do not edit these files in the 4Test Editor.
3. Update the references to the source location of your files. If the location of your `projectname.vtp` and `projectname.ini` files has changed, make sure you update that as well. Each file refers to the other.

The ProjectProfile section in the `projectname.vtp` file is required. SilkTest Classic will not be able to load your project if this section does not exist.

Troubleshooting Basic Workflow Issues

The following troubleshooting tips may help you with the basic workflow:

I restarted my application, but the Test button is not enabled

In order to enable the **Test** button on the **Test Extensions** dialog box, you must restart your application. Do not restart SilkTest Classic; restart the application that you selected on the **Enable Extensions** dialog box.

You must restart the application in the same manner. For example, if you are testing:

- A standalone Java application that you opened through a Command Prompt, make sure that you close and restart both the Java application and the Command Prompt window .
- A browser application or applet, make sure you return to the page that you selected on the **Enable Extensions** dialog box.

- An AOL browser application, make sure that you do not change the state of the application, for example resizing, or you may have issues with playback.

You can configure only one Visual Basic application at a time.

The test of my enabled Extension failed – what should I do?

If the test of your application fails, see *Troubleshooting Configuration Test Failures* for general information.

SilkTest Classic Agents

This section describes the software processes that translate the commands in your 4Test scripts into GUI-specific commands.

Overview of SilkTest Classic Agents

The SilkTest Classic 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.

SilkTest 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. The Open Agent provides the majority of the same record capabilities as the Classic Agent and the same replay capabilities.

When you create a new project, SilkTest 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, SilkTest Classic uses the Open Agent.

The Open Agent supports the following technology types:

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

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

The Classic Agent supports the following technology types:

- Java Applets
- Java AWT applications
- Java Swing applications
- Web applications
- Web with ActiveX/Visual Basic applications
- Windows API-based client/server applications

Setting the Default Agent

SilkTest Classic automatically assigns a default Agent to your project or scripts. When you create a new project, the type of project that you select determines the default Agent. For instance, if you specify that you want to create an Adobe Flex or Windows API-based client/server project, the Open Agent is

automatically set as the default Agent. When you open a project that was created with an earlier version of SilkTest, SilkTest Classic detects which Agent was used and sets it as the default Agent. At any time, you can configure SilkTest Classic to automatically use the Open Agent or the Classic Agent as the default Agent. To set the default Agent so the Agent that you use most often is automatically assigned to scripts, click the appropriate icon in the toolbar or specify the default Agent in the **Runtime Options** dialog box.

SilkTest Classic automatically starts the default Agent when you open a project or create a new project.

When you enable extensions, set the recovery system, configure the application, or record a test case, SilkTest Classic uses the default Agent. When you run a test, SilkTest Classic automatically connects to the appropriate Agent. SilkTest Classic uses the window declaration, locator, or Find or FindAll command to determine which Agent to use.

Differences in the Classes Supported by the Open and Classic Agents

Slight differences exist in the classes available for each Agent.

Functions and Methods that Use the Classic Agent Only

Certain functions and methods run on the Classic Agent only. As a result, if you are running an Open Agent project, the Classic Agent may also open because a function or method requires the Classic Agent.

To help you determine which methods are supported on each Agent, SilkTest Classic includes two 4Test keywords.

How SilkTest Classic Assigns an Agent to a Window Declaration

When you record a test with the Open Agent set as the default agent, SilkTest Classic includes a locator to identify the top-most window of the test application. For instance, this window declaration for a Notepad application that uses the Open Agent includes the following locator:

```
window MainWin UntitledNotepad
locator "/MainWin[@caption='Untitled - Notepad']"
```

SilkTest Classic determines which Agent to use by detecting whether a locator or `Find` or `FindAll` command is used. If no locator or `Find` or `FindAll` command is present, SilkTest Classic uses the Classic Agent.

In earlier releases, the `TAG_IS_OPEN_AGENT` tag was defined on the root window declaration of a control hierarchy to identify that the Open Agent should be used. This is no longer necessary. When SilkTest Classic detects a locator on the top-most window or detects a `Find` or `FindAll` command, the Open Agent is automatically used. When a window declaration contains both locators and tags and either could be used for resolving the window, check or uncheck the **Prefer Locator** check box in the **General Options** dialog box to determine which method is used.

Setting the Default Agent

SilkTest Classic automatically starts the default agent when you open a project or create a new project. You can configure SilkTest Classic to automatically connect to the Open Agent or the Classic Agent by default.

To set the default agent, you can use:

- The **Runtime Options** dialog box.
- Icons in the toolbar.

Setting the Default Agent Using the Runtime Options Dialog Box

To set the default agent using the **Runtime Options** dialog box:

1. In the main menu, click **Options** > **Runtime**.
   The **Runtime Options** dialog box opens.
2. Select the agent that you want to use as the default from the **Default Agent** list box.
3. If you use the Classic Agent, select the type of network you want to use in the **Network** list box. If you select the Open Agent, TCP/IP is automatically selected.
4. If you use named agents, select the local agent name from the **Agent Name** list box. For instance, if your environment uses multiple agents or a port that uses a value other than the default, select the local agent.
5. Click **OK**.

   When you record a test case, SilkTest Classic automatically uses the default agent.

Setting the Default Agent Using the Toolbar Icons

From the main toolbar, click the following icons to set the default agent:

-  to use the Classic Agent.
-  to use the Open Agent.

Connecting to the Default Agent

Typically, the default agent starts automatically when it is needed by SilkTest Classic. However, you can connect to the default agent manually if it does not start or to verify that it has started.

To connect to the default Agent, from the main menu, click **Tools** > **Connect to Default Agent**.

The command starts the Classic Agent or the Open Agent on the local machine, depending on which agent is specified as the default in the **Runtime Options** dialog box. If the Agent does not start within 30 seconds, a message is displayed. If the default Agent is configured to run on a remote machine, you must connect to it manually.

Creating a Script that Uses Both Agents

You can create a script that uses the Classic Agent and the Open Agent. Recording primarily depends on the default agent while replaying the script primarily depends on the window declaration of the underlying control. If you create a script that does not use window declarations, the default agent is used to replay the script.

1. Set the default Agent to the Classic Agent.
2. Enable extensions for the application automatically using the **Basic Workflow**.
3. Click **Record Testcase** in the **Basic Workflow** bar and record your test case.
4. When prompted, click **Paste to Editor** and then click **Paste testcase** and update window declaration(s).
   The frame now contains window declarations from the Classic Agent.
5. Click **File** > **Save** to save the test case.
6. Set the default Agent to the Open Agent.

7. Click **Options** > **Application Configurations**.
   The **Edit Application Configurations** dialog box opens.
8. Click **Add**.

   The **New Application Configuration** wizard opens.
9. Using the wizard, configure a standard or Web site test configuration.
10. Click **OK**.
11. Click **Record Testcase** in the **Basic Workflow** bar and record your test case.
12. When prompted, click **Paste to Editor** and then click **Paste testcase and update window declaration(s)**.
    The frame now contains window declarations from both Classic and Open Agent. SilkTest Classic automatically detects which Agent is required for each test based on the window declaration and changes the Agent accordingly.
13. Click **File** > **Save** to save the test case.
14. Click **Run Testcase** in the **Basic Workflow** bar to replay the test case.
    SilkTest Classic automatically recognizes which Agent to use based on the underlying window declarations.

You can also use the function `Connect([sMachine, sAgentType])` in a script to connect a machine explicitly with either Classic or Open Agent. Using the connect function changes the default Agent temporarily for the current test case, but it does not change the default Agent of your project. However, this does not override the Agent that is used for replay, which is defined by the window declaration.

Overview of Record Functionality Available for the SilkTest Classic Agents

The Open Agent provides the majority of the same record capabilities as the Classic Agent and the same replay capabilities.

The following table lists the record functionality available for each SilkTest Classic Agent.

Record Command	Classic Agent	Open Agent
Window Declarations	Supported	Supported
Application State	Supported	Supported
Testcase	Supported	Supported
Actions	Supported	Supported
Window Identifiers	Supported	Not Supported
Window Locations	Supported	Not Supported
Window Locators	Not Supported	Supported
Class/Scripted	Supported	Not Supported
Class/ Accessibility	Supported	Not Supported
Method	Supported	Not Supported

Record Command	Classic Agent	Open Agent
Defined Window	Supported	Not Supported

Note: SilkTest Classic determines which Agent to use by detecting whether a locator or `Find` or `FindAll` command is used. If a locator or `Find` or `FindAll` command is present, SilkTest Classic uses the Open Agent. As a result, you do not need to record window declarations for the Open Agent. For calls that use window declarations, the Agent choice is made based on the presence or absence of the locator keyword and on the presence or absence of `TAG_IS_OPEN_AGENT` in a tag or multitag. When a window declaration contains both locators and tags and either could be used for resolving the window, check or uncheck the **Prefer Locator** check box in the **General Options** dialog box to determine which method is used.

Setting Record and Replay Options for the Open Agent

You can set Agent options using the **Recording Options** dialog box or you can use `SetOption` within a script. If you use `SetOption`, it overrides the values specified in the **Recording Options** dialog box. If you do not set an option with `SetOption`, the value specified in the **Recording Options** dialog box is the default. Choose **Options** > **Recorder** to open the **Recording Options** dialog box. Using the **Recording Options** dialog box you can:

- Set recording preferences.
- Set recording options for xBrowser.
- Set custom attributes to use in locators.
- Set classes to ignore.
- Set WPF classes to expose during recording and playback.
- Set xBrowser synchronization options.
- Set replay options.

Setting the Window Timeout Value to Prevent Window Not Found Exceptions

The window timeout value is the number of seconds SilkTest Classic waits for a window to display. If the window does not display within that period, the Window not found exception is raised. For example, loading an Adobe Flex application and initializing the Adobe Flex automation framework may take some time, depending on the machine on which you are testing and the complexity of your Adobe Flex application. In this case, setting the Window timeout value to a higher value enables your application to fully load.

If you suspect that SilkTest Classic is not waiting long enough for a window to display, you can increase the window timeout value in the following ways:

- Change the window timeout value on the **Timing** tab of the **Agent Options** dialog box.
- Manually add a line to the script.

If the window is on the screen within the amount of time specified in the window timeout, the tag for the object might be the problem.

Manually Setting the Window Timeout Value

In some cases, you may want to increase the window timeout value for a specific test, rather than for all tests in general. For example, you may want to increase the timeout for Flex application tests, but not for browser tests.

1. Open the test script.
2. Add the following to the script: `Agent.SetOption (OPT_WINDOW_TIMEOUT, numberOfSeconds)`.

Setting the Window Timeout Value in the Agent Options Dialog Box

To change the window timeout value in the **Agent Options** dialog box:

1. Click **Options** > **Agent**.
2. Click the **Timing** tab.
3. Type the value into the **Window timeout** text box.

   The value should be based on the speed of the machine, on which you are testing, and the complexity of the application that you are testing. By default, this value is set to 5 seconds. For example, loading and initializing complex Flex applications generally requires more than 5 seconds.
4. Click **OK**.

Configuring SilkTest Classic Open Agent Port Numbers

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. 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.

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.

After changing the port number, restart the Open Agent, SilkTest Classic, Silk4J, SilkTest Recorder, and the application that you want to test.

Open Agent Port Numbers

When the Open Agent starts, a random, available port is assigned to SilkTest Classic, SilkTest Recorder, Silk4J, and the application that you are testing. The port numbers are registered on the information service. SilkTest Classic, SilkTest Recorder, and 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 SilkTest Classic, SilkTest Recorder, or Silk4J, connects to that port. Communication runs directly between SilkTest Classic, SilkTest Recorder, 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.

Migrating from the Classic Agent to the Open Agent

This section includes several useful topics that explain 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.

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_PAUSE_TRUELOG	TrueLog Explorer is not supported on the Open Agent.
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. SilkTest Classic automatically expands the syntax to use full XPath strings when you run a script.

You can omit:

- The hierarchy separator, “.//”. SilkTest Classic defaults to “//”.
- The class name. SilkTest 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 Agents

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>
PopupSelect	<no corresponding class>

Classic Agent	Open Agent
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>
HtmlComboBox	<no corresponding class>
HtmlForm	DomForm

Classic Agent	Open Agent
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>
JavaAwtScrollBar	AWTScrollBar
JavaAwtStaticText	AWTLabel
JavaAwtTextField	AWTTextField, AWTTextArea
JavaJFCCheckBox	JCheckBox

Classic Agent	Open Agent
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*.

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

The Classic Agent and the Open Agent differ slightly in the function parameters 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.

For some parameters, the Open Agent uses a hard-coded default value internally. If one of these parameters is set in a 4Test script, the Open Agent ignores the value and uses the value listed here.

Function	Parameter	Classic Agent Value	Open Agent Value
AnyWin::PressKeys/ ReleaseKeys	nDelay	Any number.	0
AnyWin::PressKeys/ ReleaseKeys	sKeys	More than one key is supported.	Only one key is supported. The first key is used and the remaining keys are ignored. For example `MainWin.PressKeys( "<Shift><Left>")` will only press the **Shift** key. To press both keys, specify `MainWin.PressKeys( "<Shift>") MainWin.PressKeys( "<Left >")`.
AnyWin::TypeKeys	sEvents	Keystrokes to type or mouse buttons to press.	The Open Agent supports keystrokes only.
AnyWin::GetChildren	bInvisible	TRUE or FALSE.	FALSE.
AnyWin::GetChildren	bNoTopLevel	TRUE or FALSE.	FALSE.
TextField::GetFontName	iLine	The Classic Agent recognizes this parameter.	The Open Agent ignores this parameter.
AnyWin::GetCaption	bNoStaticText	TRUE or FALSE.	FALSE.
AnyWin::GetCaption, Control::GetPriorStatic	bRawMode	TRUE or FALSE.	FALSE. However, the returned strings include trailing and leading spaces, but ellipses, accelerators, and hot keys are removed.
PageList::GetContents/ GetPageName	bRawMode	TRUE or FALSE.	FALSE. However, the returned strings include trailing and leading spaces, ellipses, and hot keys but accelerators are removed.
AnyWin::Click/ DoubleClick/ MoveMouse/ MultiClick/ PressMouse/ ReleaseMouse, PushButton::Click	bRawEvent	The Classic Agent recognizes this parameter.	The Open Agent ignores this value.

Overview of the Methods Supported by the SilkTest Classic Agents

The `winclass.inc` file includes information about which methods are supported for each SilkTest 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. SilkTest 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.

SYS Functions Supported by the Open Agent and the Classic Agent

The Classic Agent supports all SYS functions. The Open Agent supports all SYS functions with the exception of `SYS_GetMemoryInfo`. `SYS_GetMemoryInfo` defaults to the Classic Agent when a script is executed.

You can use the following SYS functions with the Open Agent or the Classic Agent.

SYS Function	Description
SYS_GetRegistryValue	With the Classic Agent, `SYS_GetRegistryValue` returns an incorrect value when a binary value is used. Use the Open Agent with `SYS_GetRegistryValue` to avoid this issue.
SYS_FileSetPointer	When setting the pointer after the end of the file, the Open Agent does not throw an exception, while the Classic Agent does throw an exception.
SYS_IniFileGetValue	The Open Agent does not allow the ‘]’ character to be part of a section name, while the Classic Agent does allow it. Also, with the Open Agent, ‘=’ must not be part of a key name. The Classic Agent allows ‘=’ to be part of a key name, but produces incorrect results.

Note: Error messages and exceptions may differ between the Open Agent and the Classic Agent.

Enabling Extensions for Applications Under Test

This section describes how you can use extensions to extend the capabilities of a program or the data that is available to the program.

Overview of Extensions

An extension is a file that serves to extend the capabilities of, or data available to, a more basic program. SilkTest Classic provides extensions for testing applications that use non-standard controls in specific development and browser environments. With the Classic Agent, SilkTest Classic provides extensions for the following environments:

- Java Applets
- Java AWT applications
- Java Swing applications
- Web with ActiveX/Visual Basic applications

Windows Forms, Windows Presentation Foundation (WPF), Java SWT/RCP, Windows-based, Web, SAP, and Adobe Flex applications are also supported and use technology domains and the Open Agent.

Enabling Extensions

Using the basic workflow, SilkTest Classic can automatically enable extensions for many different development environments.

You can also click **Tools** > **Enable Extensions** to have SilkTest Classic automatically set your extension.

If the **Basic workflow** does not support your configuration, you can enable the extension manually.

Related Files

If you are using a project, the extension configuration information is stored in the `partner.ini` file. If you are not using a project, the extension configuration information is stored in the `extend.ini` file.

When you enable extensions, SilkTest Classic adds an include file based on the technology or browser type that you enable to the **Use files** location in the **Runtime Options** dialog box. Extensions that use technologies on the Classic Agent are located in the `<SilkTest Classic project directory>\extend\` directory.

Extensions that SilkTest Classic can Automatically Configure

This functionality is available only for projects or scripts that use the Classic Agent.

Using the **Basic Workflow**, SilkTest Classic can automatically configure extensions for many development environments, including:

- Browser applications and applets running in Internet Explorer, Netscape, Firefox, and AOL
- .NET standalone Windows Forms applications
- Windows Presentation Foundation (WPF) applications
- Standalone Java and Java AWT applications

- Java Web Start applications and InstallAnywhere applications and applets
- Java SWT applications
- Visual Basic applications
- Client/Server applications
- Adobe Flex applications

You cannot enable extensions for SilkTest Classic (`partner.exe`), Classic Agent (`agent.exe`), or Open Agent (`openAgent.exe`).

For specific information about supported versions, refer to the *Release Notes*.

If the **Basic Workflow** does not support your configuration, you can enable the extension manually.

If you use the Classic Agent, the **Basic Workflow** does not automatically configure browser applications containing ActiveX objects. To configure a browser application with ActiveX objects, check the ActiveX check box in the row for the extension you are enabling in the **Extensions** dialog box. Or, test browser applications using the Open Agent.

Extensions that Must be Set Manually

This functionality is available only for projects or scripts that use the Classic Agent.

Using the **Basic Workflow**, SilkTest Classic can automatically enable extensions for many different development environments. If the **Basic Workflow** does not support your configuration or you prefer to enable extensions manually, enable the extension on your host machine and enable the extension on your target machine, regardless of whether the application you plan to test will run locally or on remote machines. Enable extensions manually if you:

- Want to change your currently enabled extension.
- Want to enable additional options for the extension you are using, such as Accessibility, Active X, or Java.
- Are testing embedded browser applications using the Classic Agent, for example, if DOM controls are embedded within a Windows Forms application.
- Are testing an application that does not have a standard name.

If you are testing Web applications using the Classic Agent, SilkTest Classic enables the extension associated with the default browser you specified on the **Select Default Browser** dialog box during the SilkTest Classic installation. If you want to use the extension you specified during the SilkTest Classic installation, you do not need to complete this procedure unless you need additional options, such as Accessibility, Java, or ActiveX.

If you are not testing Java but do have Java installed, we recommend that you disable the classpath before using SilkTest Classic.

SilkTest Classic automatically enables Java support in the browser if your web page contains an applet. The **Enable Applet Support** check box on the **Extension Settings** dialog for browser is automatically selected when the **Enable Extensions** workflow detects an applet. You can uncheck the check box to prevent SilkTest Classic from loading the extension. If no applet is detected, the check box is not available.

Extensions on Host and Target Machines

This functionality is available only for projects or scripts that use the Classic Agent.

You must define which extensions SilkTest Classic should load for each application under test, regardless of whether the application will run locally or on remote machines. You do this by enabling extensions on your host machine and on each target machine before you record or run tests.

Extensions on the host machine

On the host machine, we recommend that you enable only those extensions required for testing the current application. Extensions for all other applications should be disabled on the host to conserve memory and other system resources. By default, the installation program:

- Enables the extension for your default Web browser environment on the host machine.
- Disables extensions on the host machine for all other browser environments.
- Disables extensions for all other development environments.

When you enable an extension on the host machine, SilkTest Classic does the following:

- Adds the include file of the extension to the **Use Files** text box in the **Runtime Options** dialog box, so that the classes of the extension are available to you.
- Makes sure that the classes defined in the extension display in the **Library Browser**. SilkTest Classic does this by adding the name of the extension's help file, which is `browser.ht`, to the **Help Files For Library Browser** text box in **General Options** dialog box and recompiling the help file used by the **Library Browser**.
- Merges the property sets defined for the extension with the default property sets. The web-based property sets are in the `browser.ps` file in the `Extend` directory. The file defines the following property sets: Color, Font, Values, and Location.

Extensions on the target machine

The **Extension Enabler** dialog box is the utility that allows you to enable or disable extensions on your target machines. All information that you enter in the **Extension Enabler** is stored in the `extend.ini` file and allows the Agent to recognize the non-standard controls you want to test on target machines.

Enabling Extensions Automatically Using the Basic Workflow

An extension is a file that serves to extend the capabilities of, or data available to, a more basic program. SilkTest Classic provides extensions for testing applications that use non-standard controls in specific development and browser environments.

If you are testing a generic project that uses the Classic Agent, perform the following procedure to enable extensions:

1. Start the application or applet for which you want to enable extensions.
2. Start SilkTest Classic and make sure the basic workflow bar is visible. If it is not, click **Workflows** > **Basic** to enable it.

   If you do not see **Enable Extensions** on the workflow bar, ensure that the default agent is set to the Classic Agent.
3. If you are using SilkTest Classic projects, click **Project** and open your project or create a new project.
4. Click **Enable Extensions**.

   You cannot enable extensions for SilkTest Classic (`partner.exe`), the Classic Agent (`agent.exe`), or the Open Agent (`openAgent.exe`).
5. Select your test application from the list on the **Enable Extensions** dialog box, and then click **Select**.
6. If your test application does not display in the list, click **Refresh**. Or, you may need to add your application to this list in order to enable its extension.
7. Click **OK** on the **Extension Settings** dialog box, and then close and restart your application.
8. If you are testing an applet, the **Enable Applet Support** check box is checked by default.

9. When the **Test Extension Settings** dialog box opens, restart your application in the same way in which you opened it; for example, if you started your application by double-clicking the .exe, then restart it by double-clicking the .exe.
10. Make sure the application has finished loading, and then click **Test**.
    When the test is finished, a dialog box displays indicating that the extension has been successfully enabled and tested. You are now ready to begin testing your application or applet. If the test fails, review the troubleshooting topics.

When you enable extensions, SilkTest Classic adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

Enabling Extensions on a Host Machine Manually

This functionality is available only for projects or scripts that use the Classic Agent.

Using the **Basic workflow**, SilkTest Classic can automatically enable extensions for many different development environments. If you would rather enable the extension manually, or the basic workflow does not support your configuration, follow the steps described in this topic.

A host machine is the system that runs the SilkTest Classic software process, in which you develop, edit, compile, run, and debug 4Test scripts and test plans.

There is overhead to having more than one browser extension enabled, so you should enable only one browser extension unless you are actually testing more than one browser in an automated session.

1. Start SilkTest Classic and click **Options** > **Extensions**.
2. If you are testing a client/server project, rich internet application project, or a generic project that uses the Classic Agent, perform the following steps:
   a) On the **Extensions** dialog box, click the extension you want to enable. You may need to add your application to this list in order to enable its extension.
   b) Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate.
   c) Disable other extensions that you will not be using by selecting **Disabled** in the **Primary Extension** field. To disable a Visual Basic extension, uncheck the **ActiveX** check box for the Visual Basic application.
   d) Click **OK**.

Manually Enabling Extensions on a Target Machine

This functionality is available only for projects or scripts that use the Classic Agent.

Using the basic workflow, SilkTest Classic can automatically enable extensions for many different development environments. If you would rather enable the extension manually, or the basic workflow does not support your configuration, follow the steps described in this topic.

A target machine is a system (or systems) that runs the 4Test Agent, which is the software process that translates the commands in your scripts into GUI-specific commands, in essence, driving and monitoring your applications under test. One Agent process can run locally on the host machine, but in a networked environment, any number of Agents can run on remote machines.

If you are running local tests, that is, your target and host are the same machine, complete this procedure and enable extensions on a host machine manually.

1. Make sure that your browser is closed.
2. From the SilkTest Classic program group, choose **Extension Enabler**. To invoke the **Extension Enabler** on a remote non-Windows target machine, run `extinst.exe`, located in the directory on the target machine in which you installed the Classic Agent.

3. Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate. To get information about the files used by an extension, select an extension and click **Details**. You may need to add your application to this list in order to enable its extension.
4. Click **OK** to close the **Extension Enabler** dialog box.

   If you enable support for ActiveX in this dialog box, make sure that it is enabled in the **Extensions** dialog box as well.
5. Restart your browser, if you enabled extensions for web testing.

   Once you have set your extension(s) on your target and host machines, verify the extension settings to check your work. Be sure to consider how you want to work with borderless tables. If you are testing non-Web applications, you must disable browser extensions on your host machine. This is because the recovery system works differently when testing Web applications than when testing non-Web applications. For more information about the recovery system for testing Web applications, see Web applications and the recovery system. When you select one or both of the Internet Explorer extensions on the host machine's **Extension** dialog box, SilkTest Classic automatically picks the correct version of the host machine's Internet Explorer application in the **Runtime Options** dialog box. If the target machine's version of Internet Explorer is not the same as the host machine's, you must remember to change the target machine's version.

Enabling Extensions for Embedded Browser Applications

To test an embedded browser application, enable the Web browser as the primary extension for the application in both the **Extension Enabler** and in the **Extensions** dialog boxes. For instance, if you are testing an application with DOM controls that are embedded within a .NET application, follow these instructions to enable extensions.

1. Click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Extension Enabler** to start the **Extension Enabler**.
2. On the **Extension Enabler** dialog box, click **New**.
3. Click  to navigate to the location of the application executable.
4. Select the executable file and then click **Open**.
5. Click **OK**.
6. From the **Primary Extension** list box, select the DOM extension for the application that you added.
7. Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate.

   For example, to test a .NET application with embedded Web controls, select a browser in the **Primary Extension** list and check the **.NET** check box for the application within the grid.
8. Click **OK**.
9. Start SilkTest Classic and then choose **Options** > **Extensions**.
10. On the **Extensions** dialog box, click **New**.
11. Click  to navigate to the location of the application executable.
12. Select the executable file and then click **Open**.
13. Click **OK**.
14. From the **Primary Extension** list box, select the DOM extension for the application that you added.
15. Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate.

    For example, to test a .NET application with embedded Web controls, select a browser in the **Primary Extension** list and check the **.NET** check box for the application within the grid.
16. Click **OK**.
17. Re-start SilkTest Classic.

The IE DOM extension may not detect changes to a web page that occur when JavaScript replaces a set of elements with another set of elements without changing the total number of elements. To force the DOM extension to detect changes in this situation, call the `FlushCache()` method on the top-level browserchild for the embedded browser.

Note: This problem may occur more often for embedded browsers than for browser pages, because SilkTest is not notified of as many browser events for embedded browsers.

Also call `FlushCache()` if you get a `Coordinate out of bounds` exception when calling a method, for example `Click()`, on an object that previously had been scrolled into view.

Note: The `BrowserPage` window identifier is not valid when using embedded browsers because the default browser type is '(none)' (NULL).

Enabling Extensions for HTML Applications (HTAs)

This functionality is available only for projects or scripts that use the Classic Agent.

You must enable extensions on the host and target machines manually in order to use HTML applications (HTAs).

Before you begin, create a project that uses the Classic Agent.

1. Click **Options** > **Extensions** to open the **Extensions** dialog box.
2. Click **New** to open the **Extension Application** dialog box.
3. Click  to navigate to the location of the `.hta` file that you want to enable. If the file name contains spaces, be sure to enclose the name in quotation marks.
4. Select the `.hta` file and then click **Open**.
5. Click **OK**.
6. In the **Primary Extension** column next to the `.hta` application that you just enabled, select `Internet Explorer`.
7. Click **OK**.
8. Start the **Extension Enabler** by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Extension Enabler**. (Or use the command line to launch `"C:\Progam Files\Silk\SilkTest \extinst.exe"`.)
9. On the **Extension Enabler** dialog box, click **New** to open the **Extension Application** dialog box.
10. Click  to navigate to the location of the `.hta` file that you want to enable. If the file name contains spaces, be sure to enclose the name in quotation marks.
11. Select the `.hta` file and then click **Open**.
12. Click **OK**.
13. In the **Primary Extension** column next to the `.hta` application that you just enabled, select `Internet Explorer`.
14. Click **OK**.

Adding a Test Application to the Extension Dialog Boxes

This functionality is available only for projects or scripts that use the Classic Agent.

You must manually add the following applications to the **Extensions** dialog box and the **Extension Enabler** dialog box:

- Applications that are embedded in Web pages and use the Classic Agent.

- All test applications that do not have standard names and use the Classic Agent.
- When you add a test application to the **Extensions** dialog box on the host machine, you should immediately add it to the **Extension Enabler** dialog box on each target machine on which you intend to test the application.

You may also add new applications by duplicating existing applications and then changing the application name.

To add a test application to the **Extension** dialog boxes:

1. Click **Options** > **Extensions** to open the **Extensions** dialog box, or open the **Extension Enabler** dialog box from the SilkTest program group.
2. If you are testing a client/server project, Rich Internet Application project, or a generic project that uses the Classic Agent, perform the following steps:
   a) Click **New** to open the **Extension Application** dialog box.
   b) Click **...** to browse to the application's executable or DLL file.

      Separate multiple application names with commas. If the executable name contains spaces, be sure to enclose the name in quotation marks.
   c) Select the executable file and then click **Open**.
   d) Click **OK**.
3. Click **OK** to close the dialog box.

Verifying Extension Settings

This functionality is available only for projects or scripts that use the Classic Agent.

If the extension settings for the host and target machines do not match, neither extension will load properly.

- To see the target machine setting, choose **Options** > **Extensions**. Verify that the Primary Extension is enabled and other extensions are enabled, if appropriate. If you enabled a browser extension, you can also verify the extension settings on the target machine by starting the browser and SilkTest Classic, and then right-clicking the task bar Agent icon and selecting **Extensions** > **Detail**.
- To verify that the setting on the host machine is correct, choose **Options** > **Runtime**. Make sure that the default browser in the **Default Browser** field on the **Runtime Options** dialog box is correct.

Why Applications do not have Standard Names

This functionality is available only for projects or scripts that use the Classic Agent.

In the following situations applications might not have standard names, in which case you must add them to the **Extension Enabler** dialog box and the **Extensions** dialog box:

- Visual Basic applications can have any name, and therefore the SilkTest Classic installation program cannot add them to the dialog box automatically.
- You are running an application developed in Java as a stand-alone application, outside of its normal runtime environment.
- You have explicitly changed the name of a Java application.

Duplicating a Test Applications Settings in Another Application

This functionality is available only for projects or scripts that use the Classic Agent.

You can add new applications to the **Extension Enabler** dialog box or the **Extensions** dialog box by duplicating existing applications and renaming the new application. All the settings of the original

application, that is, primary extension, other extensions, or options set on the **Extensions** dialog box, are copied.

You can only duplicate applications that you entered manually and that use the Classic Agent.

To copy a test application's settings into another application:

1. Click **Options** > **Extensions** to open the **Extensions** dialog box, or open the **Extension Enabler** dialog box from the SilkTest Classic program group.
2. Select the application that you want to copy.
3. Click **Duplicate**.
   The **Extension Application** dialog box opens.
4. Type the name of the new application you want to copy.

   Separate multiple application names with commas.
5. Click **OK** to close the **Extension Application** dialog box.
   The new applications display in the dialog box you opened.
6. Click **OK** to close the dialog box.

Deleting an Application from the Extension Enabler or Extensions Dialog Box

This functionality is available only for projects or scripts that use the Classic Agent.

After completing your testing of an application or if you make a mistake, you might want to delete the application from the **Extension Enabler** dialog box or the **Extensions** dialog box. You can delete only applications that you have entered manually. Visual Basic applications fall into this category.

To remove an application from the **Extension Enabler** or **Extensions** dialog box:

1. Click **Options** > **Extensions** to open the **Extensions** dialog box, or open the **Extension Enabler** dialog box from the SilkTest Classic program group.
2. Select the application that you want to delete from the dialog box.
3. Click **Remove**.
   The application name is removed from the dialog box.
4. Click **OK**.

Disabling Browser Extensions

This functionality is available only for projects or scripts that use the Classic Agent.

1. In SilkTest Classic, choose **Options** > **Extensions**.
2. From the **Primary Extension** list, select **Disabled** for the extension you want to disable.
3. In the **Other extensions** field, uncheck any checked check boxes.
4. Click **OK**.

If you are testing non-Web applications, you must disable browser extensions on your host machine. This is because the recovery system works differently when testing Web applications than when testing non-Web applications.

Comparison of the Extensions Dialog Box and the Extension Enabler Dialog Box

This functionality is available only for projects or scripts that use the Classic Agent.

The **Extensions** dialog box and the **Extension Enabler** dialog box look similar; they are both based on a grid and have identical column headings and have some of the same buttons. However, they configure different aspects of the product:

	Extensions Dialog Box	Extension Enabler Dialog Box
Enables AUTs and extensions	On host machine	On target machines
Provides information for	SilkTest Classic	Agent
Available from	**Options** menu	SilkTest Classic program group
Information stored in	`partner.ini`	`extend.ini`
When to enable/disable AUTs and extensions	Enable the AUTs and extensions you want to test now; disable others.	Enable all AUTs and extensions you ever intend to test. No harm in leaving them enabled, even if you are not testing them now.
What you specify on each: • Primary environment • Java or ActiveX, if required • Accessibility	• Yes, according to the type • Enable and set options • Enable and set options	• Yes, according to the type • Enable only • Enable only
What installation does: • Default browser (If any) • Other browsers (if any) • Java runtime environment • Oracle Forms runtime environment • Visual Basic 5 & 6	• Displayed and enabled • Displayed but disabled • Displayed but disabled • Displayed but disabled • Not displayed or enabled	• Displayed and enabled • Displayed and enabled • Displayed and enabled • Displayed but disabled • Not displayed or enabled

Configuring the Browser

In order for SilkTest Classic to work properly, make sure that your browser is configured correctly.

If your tests use the recovery system of SilkTest Classic, that is, your tests are based on DefaultBaseState or on an application state that is ultimately based on DefaultBaseState, SilkTest Classic makes sure that your browser is configured correctly.

If your tests do not use the recovery system, you must manually configure your browser to make sure that your browser displays the following items:

- The standard toolbar buttons, for example **Home**, **Back**, and **Stop**, with the button text showing. If you customize your toolbars, then you must display at least the **Stop** button.
- The text box where you specify URLs. **Location** in Netscape; **Address** in Explorer.
- Links as underlined text.
- The browser window's menu bar in your Web application. It is possible through some development tools to hide the browser window's menu bar in a Web application. SilkTest Classic will not work properly unless the menu bar is displayed. The recovery system cannot restore the menu bar, so you must make sure the menu bar is displayed.
- The status bar at the bottom of the window shows the full URL when your mouse pointer is over a link.

We recommend that you configure your browser to update cached pages on a frequent basis.

Internet Explorer

1. Click **Tools** > **Internet Options**, then click the **General** tab.
2. In the **Temporary Internet Files** area, click **Settings**.
3. On the **Settings** dialog box, select **Every visit to the page** for the **Check for newer versions of stored pages** setting.

Firefox

1. Choose **Edit** > **Preferences** > **Advanced** > **Cache**.
2. Indicate when you want to compare files and update the cache. Select **Every time I view the page** at the **Compare the page in the cache to the page on the network** field.

AOL

Even though AOL's Proxy cache is updated every 24 hours, you can clear the AOL Browser Cache and force a page to reload. To do this, perform one of the following steps:

- Delete the files in the temporary internet files folder located in the Windows directory.
- Press the **CTRL** key on your keyboard and click the AOL browser reload icon (Windows PC only).

Friendly URLs

Some browsers allow you to display "friendly URLs," which are relative to the current page. To make sure you are not displaying these relative URLs, in your browser, display a page of a web site and move your mouse pointer over a link in the page.

- If the status bar displays the full URL (one that begins with the http:// protocol name and contains the site location and path), the settings are fine. For example: `http://www.mycompany.com/products.htm`
- If the status bar displays only part of the URL (for example, `products.htm`), turn off "friendly URLs." (In Internet Explorer, this setting is on the **Advanced** tab of the **Internet Options** dialog box.)

Setting Agent Options for Web Testing

When you first install SilkTest Classic, all the options for web testing are set appropriately. If, for some reason (perhaps you were testing non-web applications and changed them), you have problems testing web applications, do the following:

1. Click **Options** > **Agent**.
2. In the **Agent Options** dialog box, check to make sure you have the following settings, and then click **OK** to close the dialog box.

Tab	Option	Specifies	Setting
Timing	`OPT_APPREADY_TIMEOUT`	The number of seconds the Agent for an application to become ready. (Browser extensions support this option.)	Site-specific; default is 180 seconds
Timing	`OPT_APPREADY_RETRY`	The number of seconds the Agent waits between attempts to verify that the application is ready.	Site-specific; default is 0.1 seconds
Other	`OPT_SCROLL_INTO_VIEW`	That the Agent scrolls a control into view before recording events against it.	TRUE (checked); default is TRUE

Tab	Option	Specifies	Setting
Other	OPT_SHOW_OUT_OF_VIEW	Allowing SilkTest to see objects not currently scrolled into view.	TRUE (checked); default is TRUE
Verification	OPT_VERIFY_APPREADY	Verify than an application is ready (browser extensions support this option)	TRUE (checked); default is TRUE

3. Click **OK**.

Specifying a Browser for SilkTest Classic to Use in Testing a Web Application

You can specify a browser for SilkTest Classic to use when testing a Web application at runtime or you can use the browser specified through the **Runtime Options** dialog box.

To completely automate your testing, consider specifying the browser at runtime. Do this in either of the following ways:

- Use the `SetBrowserType` function in a script. This function takes an argument of type `BROWSERTYPE.`
- Pass an argument of type `BROWSERTYPE` to a test case as the first argument.

For an example of passing browser specifiers to a test case, see the second example in `BROWSERTYPE`. It shows you how to automate the process of running a test case against multiple browsers.

Specifying a browser through the Runtime Options dialog box

When you run a test and do not explicitly specify a browser, SilkTest Classic uses the browser specified in **Runtime Options** dialog box. To change the browser type, you can:

1. Run a series of tests with a specific browser.
2. Specify a different browser in the **Runtime Options** dialog box.
3. Run the tests again with the new browser.

Most tests will run unchanged between browsers.

Specifying your Default Browser

Whenever you record and run test cases, you must specify a default browser so SilkTest Classic knows which browser to use. If you did not choose a default browser during SilkTest Classic installation or if want to change the default browser, do the following:

1. Click **Options** > **Runtime**.
2. Select the browser you want to use from the **Default Browser** list box in the **Runtime Options** dialog. The list box displays browsers whose extensions you have enabled.
3. Click **OK**.

Understanding the Recovery System

This section describes how you can run unattended tests with the built-in recovery system.

Overview of the Recovery System

The built-in recovery system is one of the most powerful features of SilkTest Classic because it allows you to run tests unattended. When your application fails, the recovery system restores the application to a stable state, known as the BaseState, so that the rest of your tests can continue to run unattended.

The recovery system can restore your application to its BaseState at any point during test case execution:

- Before the first line of your test case begins running, the recovery system restores the application to the BaseState even if an unexpected event corrupted the application between test cases.
- During a test case, if an application error occurs, the recovery system terminates the execution of the test case, writes a message in the error log, and restores the application to the BaseState before running the next test case.
- After the test case completes, if the test case was not able to clean up after itself, for example it could not close a dialog box it opened, the recovery system restores the application to the BaseState.
- The recovery system cannot recover from an application crash that produces a modal dialog box, such as a **General Protection Fault (GPF)**.

SilkTest Classic uses the recovery system for all test cases that are based on DefaultBaseState or based on a chain of application states that ultimately are based on DefaultBaseState.

- If your test case is based on an application state of none or a chain of application states ultimately based on none, all functions within the recovery system are not called. For example, SetAppState and SetBaseState are not called, while DefaultTestCaseEnter, DefaultTestCaseExit, and error handling are called.

Such a test case will be defined in the script file as:

```
testcase Name () appstate none
```

SilkTest Classic records test cases based on DefaultBaseState as:

```
testcase Name ()
```

How the default recovery system is implemented

The default recovery system is implemented through several functions.

Function	Purpose
`DefaultBaseState`	Restores the default BaseState, then call the application's BaseState function, if defined.
`DefaultScriptEnter`	Executed when a script file is first accessed. Default action: none.
`DefaultScriptExit`	Executed when a script file is exited. Default action: Call the ExceptLog function if the script had errors.
`DefaultTestCaseEnter`	Executed when a test case is about to start. Default action: Set the application state.

Function	Purpose
`DefaultTestCaseExit`	Executed when a test case has ended. Default action: Call the ExceptLog function if the script had errors, then set the BaseState.
`DefaultTestPlanEnter`	Executed when a test plan is entered. Default action: none.
`DefaultTestPlanExit`	Executed when a test plan is exited. Default action: none.

You can write functions that override some of the default behavior of the recovery system.

Setting the Recovery System for the Open Agent

The recovery system ensures that each test case begins and ends with the application in its intended state. SilkTest Classic refers to this intended application state as the BaseState. The recovery system allows you to run tests unattended. When your application fails, the recovery system restores the application to the BaseState, so that the rest of your tests can continue to run unattended.

For applications that use the Open Agent and dynamic object recognition, the recovery system is configured automatically whenever the **New frame file** dialog box opens and you save a file. This dialog box opens when:

- You click **Configure Applications** on the **Basic Workflow** bar and follow the steps in the wizard.
- You click **File** > **New** and click **Test frame**.
- You click the **Create a new file** icon in the toolbar and then click **Test frame**.
- You click **Record** > **Testcase**, **Record** > **Application State**, or **Record** > **Window Locators** before you configure an application, the **New Test Frame** dialog box opens before recording starts.

If you are testing an application that uses both the Classic Agent and the Open Agent, set the Agent that will start the application as the default Agent and then set the recovery system. If you use the Classic Agent to start the application, set the recovery system for the Classic Agent.

Setting the Recovery System for the Classic Agent

The recovery system ensures that each test case begins and ends with the application in its intended state. SilkTest Classic refers to this intended application state as the BaseState. The recovery system allows you to run tests unattended. When your application fails, the recovery system restores the application to the BaseState, so that the rest of your tests can continue to run unattended.

If you are testing an application that uses both the Classic Agent and the Open Agent, set the Agent that will start the application as the default Agent and then set the recovery system. If you use the Open Agent to start the application, set the recovery system for the Open Agent.

1. Make sure the application that you are testing is running.
2. Click **Set Recovery System** on the **Basic Workflow** bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
3. From the **Application** list, click the name of the application that you are testing.

   All open applications that are not minimized are listed. This list is dynamic and will update if you open a new application. If you are connected to the Open Agent, only those applications that have extensions enabled display in the list.

Note: If you selected a non-web application as the application:

- The **Command line** text box displays the path to the executable (`.exe`) for the application that you have selected.
- The **Working directory** text box displays the path of the application you selected.

If you selected a web application, the **Start testing on this page** text box displays the URL for the application you selected. If an application displays in the list, but the URL does not display in this text box, your extensions may not be enabled correctly. Click **Enable Extensions** in the **Basic Workflow** bar to automatically enable and test extension settings.

4. *Optional:* In the **Frame file name** text box, modify the frame file name and click **Browse** to specify the location in which you want to save this file.

   Frame files must have a .inc extension. By default, this field displays the default name and path of the frame file you are creating. The default is `frame.inc`. If `frame.inc` already exists, SilkTest appends the next logical number to the new frame file name; for example, `frame1.inc`.
5. *Optional:* In the **Window name** text box, change the window name to use a short name to identify your application.
6. Click **OK**.
7. Click **OK** when the message indicating that the recovery system is configured displays.
8. A new 4Test include file, `frame.inc`, opens in the **SilkTest Editor**. Click the plus sign in the file to see the contents of the frame file.
9. Record a test case.

Base State

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

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

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

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

When an error occurs, SilkTest Classic does the following:

- Stops execution of the test case.
- Transfers control to the recovery system, which restores the application to its base state and logs the error in a results file.
- Resumes script execution by running the next test case after the failed test case.

The recovery system makes sure that the test case was able to "clean up" after itself, so that the next test case runs under valid conditions.

DefaultBaseState Function

SilkTest Classic provides a `DefaultBaseState` for applications, which ensures the following conditions are met before recording and executing a test case:

- The application is running.
- The application is not minimized.

- The application is the active application.
- No windows other than the application's main window are open. If the UI of the application is localized, you need to replace the strings, which are used to close a window, with the localized strings. The preferred way to replace these buttons is with the *lsCloseWindowButtons* variable in the object's declaration. You can also replace the strings in the **Close** tab of the **Agent Options** dialog box.

For Web applications that use the Open Agent, the `DefaultBaseState` also ensures the following for browsers, in addition to the general conditions listed above:

- The browser is running.
- Only one browser tab is open, if the browser supports tabs and the frame file does not specify otherwise.
- The active tab is navigated to the URL that is specified in the frame file.

For web applications that use the Classic Agent, the `DefaultBaseState` also ensures the following for browsers, in addition to the general conditions listed above:

- The browser is ready.
- Constants are set.
- The browser has toolbars, location and status bar are displayed.
- Only one tab is opened, if the browser supports tabs.

DefaultBaseState Types

SilkTest Classic includes two slightly different base state types depending on whether you use the Open Agent and dynamic object recognition or traditional hierarchical object recognition. When you use dynamic object recognition, SilkTest Classic creates a window object named `wDynamicMainWindow` in the base state. When you set the recovery system for a test that uses hierarchical object recognition, SilkTest Classic creates a window object called `wMainWindow` in the base state. SilkTest Classic uses the window object to determine which type of `DefaultBaseState` to execute.

Adding Tests that Use the Open Agent to the DefaultBaseState

If you want the recovery system to perform additional steps after it restores the default base state, record a new test case based on no application state and paste it into the declaration for your application's main window.

1. Open your application and the application's test frame file.
2. Click **Record** > **Testcase**.
   SilkTest Classic displays the **Record Testcase** dialog box.
3. From the **Application state** list box, click `(None)`.
4. Click **Start Recording**.
   SilkTest Classic opens the **Recording** window, which indicates that you can begin recording the `BaseState` method.
5. When you have finished recording the `BaseState` method, click **Stop Recording** on the **Recording** window.
   SilkTest Classic redisplays the **Record Testcase** dialog box.
6. Click **Paste to Editor** and then copy and paste the script in the declaration for your main window in the test frame file.
7. Choose **File** > **Save** to save the test frame file.

Adding Tests that Use the Classic Agent to the DefaultBaseState

If you want the recovery system to perform additional steps after it restores the default base state, record a new method named `BaseState` and paste it into the declaration for your application's main window. SilkTest Classic provides the Record/Method menu command to record a `BaseState` method.

1. Open your application and the application's test frame file.
2. Place the insertion point on the declaration for the application's main window.
3. Click **Record** > **Method**.
   SilkTest Classic displays the **Record Method** dialog box, which allows you to record a method for a class or window declaration.
4. From the **Method Name** list box, select `BaseState`.
5. Click **Start Recording**.
   SilkTest Classic closes the **Record Method** dialog box and displays the **Record Status** window, which indicates that you can begin recording the `BaseState` method. The Status field flashes the word `Recording`.
6. When you have finished recording the `BaseState` method, click **Done** on the **Record Status** window. SilkTest Classic redisplays the **Record Method** dialog box. The **Method Code** field contains the 4Test code you recorded.
7. Click **OK** to close the **Record Method** dialog box and place the new `BaseState` method in the declaration for your main window.

DefaultBaseState and the wDynamicMainWindow Object

SilkTest Classic executes the `DefaultBaseState` for dynamic object recognition when the default agent is the Open Agent and the global constant `wDynamicMainWindow` is defined. `DefaultBaseState` works with the `wDynamicMainWindow` object in the following ways:

1. If the `wDynamicMainWindow` object does not exist, invoke it, either using the Invoke method defined for the `MainWin` class or a user-defined Invoke method built into the object.
2. If the `wDynamicMainWindow` object is minimized, restore it.
3. If there are child objects of the `wDynamicMainWindow` open, close them.
4. If the `wDynamicMainWindow` object is not active, make it active.
5. If there is a `BaseState` method defined for the `wDynamicMainWindow` object, execute it.

DefaultBaseState and wMainWindow

SilkTest Classic executes the `DefaultBaseState` for hierarchical object recognition when the global constant `wMainWindow` is defined. `DefaultBaseState` works with the `wMainWindow` object in the following ways:

1. If the `wMainWindow` object does not exist, invoke it, either using the `Invoke` method defined for the `MainWin` class or a user-defined `Invoke` method built into the object. If `wMainWindow` is a `BrowserChild` object and the browser is not loaded, load the browser before loading the web page into it.
2. If the `wMainWindow` object is minimized, restore it. If `wMainWindow` is a `BrowserChild` object and the browser is minimized, restore it.

3. If there are child objects of the `wMainWindow` open, close them. If `wMainWindow` is a `BrowserChild` object, close any children of the browser.
4. If the `wMainWindow` object is not active, make it active.
5. If there is a `BaseState` method defined for the `wMainWindow` object, execute it.

In a scenario where a dialog box or a different Web page loads on request, the recovery system expects that web page to be loaded. However, it might not be if a Login page loads first. You can configure SilkTest Classic to handle login pages.

Flow of Control

This section describes the flow of control during the execution of each of your test cases.

The Non-Web Recovery Systems Flow of Control

Before you modify the recovery system, you need to understand the flow of control during the execution of each of your test cases. The recovery system executes the `DefaultTestcaseEnter` function. This function, in turn, calls the `SetAppState` function, which does the following:

1. Executes the test case.
2. Executes the `DefaultTestcaseExit` function, which calls the `SetBaseState` function, which calls the lowest level application state, which is either the `DefaultBaseState` or any user defined application state.

   Note: If the test case uses `AppState` none, the `SetBaseState` function is not called.

`DefaultTestCaseEnter`() is considered part of the test case, but `DefaultTestCaseExit`() is not. Instead, `DefaultTestCaseExit`() is considered part of the function that runs the test case, which implicitly is `main`() if the test case is run standalone. Therefore an unhandled exception that occurs during `DefaultTestCaseEnter`() will abort the current test case, but the next test case will run. However, if the exception occurs during `DefaultTestCaseExit`(), then it is occurring in the function that is calling the test case, and the function itself will abort. Since an application state may be called from both `TestCaseEnter`() and `TestCaseExit`(), an unhandled exception within the application state may cause different behavior depending on whether the exception occurs upon entering or exiting the test case.

Web Applications and the Recovery System

This functionality is available only for projects or scripts that use the Classic Agent.

When the recovery system needs to restore the base state of a Web application that uses the Classic Agent, it does the following:

1. Invokes the default browser if it is not running.
2. Restores the browser if it is minimized.
3. Closes any open additional browser instances or message boxes.
4. Makes sure the browser is active and is not loading a page.
5. Sets up the browser as required by SilkTest Classic.

   The recovery system performs the next four steps only if the `wMainWindow` constant is set and points to the home page in your application.
6. If `bDefaultFont` is defined and set to TRUE for the home page, sets the fonts.
7. If `BrowserSize` is defined and set for the home page, sets the size of the browser window.
8. If `sLocation` is defined and set for the home page, loads the page specified by `sLocation`.

9. If `wMainWindow` defines a `BaseState` method, executes it.
10. For additional information, see *DefaultBaseState and the wMainWindow Object*.

To use the recovery system, you must have specified your default browser in the **Runtime Options** dialog box. If the default browser is not set, the recovery system is disabled. There is one exception to this rule: You can pass a browser specifier as the first argument to a test case. This sets the default browser at runtime. For more information, see *BROWSERTYPE Data Type*.

The constant `wMainWindow` must be defined and set to the identifier of the home page in the Web application for the recovery system to restore the browser to your application's main page. This window must be of class `BrowserChild`. When you record a test frame, the constant is automatically defined and set appropriately. If you want, you can also define a `BaseState` method for the window to execute additional code for the base state, for example if the home page has a form, you might want to reset the form in the `BaseState` method, so that it will be empty at your base state.

On Windows Internet Explorer 7.x and 8.x, when recording a new frame file using **Set Recovery System**, by default SilkTest Classic does not explicitly state that the parent of the window is a browser. To resolve this issue, add the "parent Browser" line to the frame file.

How the Non-Web Recovery System Closes Windows

The built-in recovery system restores the base state by making sure that the non-Web application is running, is not minimized, is active, and has no open windows except for the main window. To ensure that only the main window is open, the recovery system attempts to close all other open windows, using an internal procedure that you can customize as you see fit.

To make sure that there are no application windows open except the main window, the recovery system calls the built-in `CloseWindows` method. This method starts with the currently active window and attempts to close it using the sequence of steps below, stopping when the window closes.

1. If a `Close` method is defined for the window, call it.
2. Click the **Close** menu item on the system menu, on platforms and windows that have system menus.
3. Click the window's close box, if one exists.
4. If the window is a dialog box, type each of the keys specified by the `OPT_ CLOSE_DIALOG_KEYS` option and wait one second for the dialog box to close. By default, this option specifies the **Esc** key.
5. If there is a single button in the window, click that button.
6. Click each of the buttons specified by the `OPT_CLOSE_WINDOW_ BUTTONS` option. By default, this option specifies the **Cancel**, **Close**, **Exit**, and **Done** keys.
7. Select each of the menu items specified by the `OPT_CLOSE_WINDOW_ MENUS` option. By default, this option specifies the **File** > **Exit** and the **File** > **Quit** menu items.
8. If the closing of a window causes a confirmation dialog box to open, `CloseWindows` attempts to close the dialog box by clicking each of the buttons specified with the `OPT_CLOSE_CONFIRM_BUTTONS` option. By default, this option specifies the **No** button.

When the window, and any resulting confirmation dialog box, closes, `CloseWindows` repeats the preceding sequence of steps with the next window, until all windows are closed.

If any of the steps fails, none of the following steps is executed and the recovery system raises an exception. You may specify new window closing procedures.

In a Web application, you are usually loading new pages into the same browser, not closing a page before opening a new one.

How the Non-Web Recovery System Starts the Application

To start a non-Web application, the recovery system executes the `Invoke` method for the main window of the application. The `Invoke` method relies on the `sCmdLine` constant as recorded for the main window when you create a test frame.

For example, here is how a declaration for the `sCmdLine` constant might look for a sample Text Editor application running under Windows:

```
const sCmdLine = "c:\ProgramFiles\<SilkTest install directory>\SilkTest
\TextEdit.exe"
```

After it starts the application, the recovery system checks whether the main window is minimized, and, if it is, uses the `Restore` method to open the icon and restore the application to its proper size.

The limit on the `sCmdLine` constant is 8191 characters.

Modifying the Default Recovery System

The default recovery system is implemented in `defaults.inc`, which is located in the directory in which you installed SilkTest Classic. If you want to modify the default recovery system, instead of overriding some of its features, as described in Overriding the default recovery system, you can modify `defaults.inc`.

We cannot provide support for modifying `defaults.inc` or the results. We recommend that you do not modify `defaults.inc`. This file might change from version to version. As a result, if you manually modify `defaults.inc`, you will encounter issues when upgrading to a new version of SilkTest Classic.

If you decide to modify `defaults.inc`, be sure that you:

- Make a backup copy of the shipped `defaults.inc` file.
- Tell Technical Support when reporting problems that you have modified the default recovery system.

Overriding the Default Recovery System

The default recovery system specifies what SilkTest Classic does to restore the base state of your application. It also specifies what SilkTest Classic does whenever:

- A script file is first accessed.
- A script file is exited.
- A test case is about to begin.
- A test case is about to exit.

You can write functions that override some of the default behavior of the recovery system.

To override	Define the following
`DefaultScriptEnter`	`ScriptEnter`
`DefaultScriptExit`	`ScriptExit`
`DefaultTestCaseEnter`	`TestCaseEnter`
`DefaultTestCaseExit`	`TestCaseExit`
`DefaultTestPlanEnter`	`TestPlanEnter`
`DefaultTestPlanExit`	`TestPlanExit`

If `ScriptEnter`, `ScriptExit`, `TestcaseEnter`, `TestcaseExit`, `TestPlanEnter`, or `TestPlanExit` are defined, SilkTest Classic uses them instead of the corresponding default function. For example, you might want to specify that certain test files are copied from a server in preparation for running a script. You might specify such processing in a function called `ScriptEnter` in your test frame.

If you want to modify the default recovery system, instead of overriding some of its features, you can modify `defaults.inc`. We do not recommend modifying `defaults.inc` and cannot provide support for modifying `defaults.inc` or the results.

Example

If you are planning on overriding the recovery system, you need to write your own `TestCaseExit(Boolean bException)`. In the following example, `DefaultTestcaseExit()` is called inside `TestCaseExit()` to perform standard recovery systems steps and the `bException` argument is passed into `DefaultTestCaseExit()`.

```
if (bException)
  DefaultTestcaseExit(bException)
```

If you are not planning to call `DefaultTestcaseExit()` and plan to handle the error logging in your own way, then you can use the `TestcaseExit()` signature without any arguments.

Use the following function signature if you plan on calling `DefaultTestCaseExit(Boolean bException)` or if your logic depends on whether an exception occurred. Otherwise, you can simply use the function signature of `TestcaseExit()` without any arguments. For example, the following is from the description of the `ExceptLog()` function.

```
TestCaseExit (BOOLEAN bException)
if (bException)
  ExceptLog()
```

Here, `DefaultTestcaseExit()` is not called, but the value of `bException` is used to determine if an error occurred during the test case execution.

Handling Login Windows

SilkTest Classic handles login windows differently, depending on whether you are testing Web or client/server applications. These topics provide information on how to handle login windows in your application under test.

Handling Login Windows in Web Applications that Use the Classic Agent

This procedure describes how to handle web applications with different possible startup pages or dialog box objects that use the Classic Agent. For example,

- A Web application requires the user to login the first time he or she visits the site in a day (a non-persistent cookie). If the user has already logged in for this browser session, the user will not be prompted for user name and password again, as the "cookie" is still available with their authorization. This could be either a login web page or a dialog box.
- A dialog box that sometimes gives a "tip of the day" or reminds the user to perform some action because it is a certain date.
- A dialog box might popup asking the user whether it is okay to download a certificate, a Java module, or some other component.

In cases such as these, you can use the *sLocation* data-member from the `wMainWindow` object as a property. You can create a property and it will look exactly like a data-member and will be called like a data-member. When trying to retrieve information from a property the `Get` portion of the property is executed. And you can add code to deal with login Web pages here.

Here are the steps of what will happen when `DefaultBaseState` runs:

1. `DefaultBaseState` will try to retrieve the *sLocation* data-member and, as such, will execute the `Get` function.
2. The `Get` function that is part of the property will actually load the web page by putting the URL of the page into the **Location** comboBox that is part of the browser and pressing `Enter`. It will then wait for the browser to report to SilkTest Classic a ready state.
3. If the **Login** page exists rather than the page we were expecting, the user name and password will be entered and the `HtmlPushButton` **OK** will be clicked. Again, SilkTest Classic will wait for the browser to return to a ready state.
4. The `Get` function returns a NULL even though at the definition of the `Get` function it was specified that a STRING would be returned. If you were to return the URL, `DefaultBaseState` would load the page again. Of course, since we have already dealt with login, it would work this time, but would add some more time into the process by loading the page again.
5. Although `DefaultBaseState` will not try to load the page, it will find it there and continue with the other steps of closing any open windows and setting the browser and Web page active.

   You can also handle unexpected and occasional dialog boxes in this way, by changing the *sLocation* data-member to a property and handling different possibilities through a Get function that is part of the property, or you can re-write the Close method. For expected security or login dialog boxes, you can set the *sUsername* and *sPassword* for the `wMainWindow` object.

```
Window BrowserChild RealPage
const PAGE_URL = http://www.somepage.com
property sLocation
STRING Get ( )

// actually load the page
Browser.SetActive ( )
Browser.Location.SetText (PAGE_URL)
Browser.Location.TypeKeys ("<Enter>")

// wait for the browser to be "ready" Browser.WaitForReady ( )
// if the Login page has shown up…
if Login.Exists ( )

// deal with it
Login.UserName.SetText (USERNAME)
Login.Password.SetText (PASSWORD)
Login.OK.Click ( )

// now wait for the browser to be ready
Browser.WaitForReady ( )

//this way DefaultBaseState will not try to load the page again
return NULL
```

Handling Login Windows in Non-Web Applications that Use the Classic Agent

Although a non-Web application's main window is usually displayed first, it is also common for a login or security window to be displayed before the main window.

Use the `wStartup` constant and the `Invoke` method

To handle login windows, record a declaration for the login window, set the value of the `wStartup` constant, and write a new `Invoke` method for the main window that enters the appropriate information into the login window and dismisses it. This enables the `DefaultBaseState` routine to perform the actions necessary to get past the login window.

You do not need to use this procedure for splash screens, which disappear on their own.

1. Open the login window that precedes the application's main window.
2. Open the test frame.
3. Click **Record** > **Window Declarations** to record a declaration for the window.
4. Point to the title bar of the window and then press **Ctrl+Alt**.
   The declaration is captured in the **Record Window Declarations** dialog box.
5. Click **Paste to Editor** to paste the declaration into the test frame.
6. In the **Record Window Declarations** dialog box, click **Close**.
7. Close your application.
8. In your test frame file, find the stub of the declaration for the `wStartup` constant, located at the top of the declaration for the main window:
   ```
   // First window to appear when application is invoked
   // const wStartup = ?
   ```
9. Complete the declaration for the `wStartup` constant by:
   - Removing the comment characters, the two forward slash characters, at the beginning of the declaration.
   - Replacing the question mark with the identifier of the login window, as recorded in the window declaration for the login window.
10. Click the `wStartup` constant and then click **Record** > **Method**.
11. On the **Record Method** dialog box, from the **Method Name** list box, select **Invoke**.
12. Open your application, but do not dismiss the login window.
13. Click **Start Recording**.
    SilkTest Classic is minimized and your application and the **SilkTest Record Status** dialog box open.
14. Perform and the record the actions that you require.
15. On the **SilkTest Record Status** dialog box, click **Done**.
    The **Record Method** dialog box opens with the actions you recorded translated into 4Test statements.
16. On the **Record Method** dialog box, click **OK** to paste the code into your include file.
17. Edit the 4Test statements that were recorded, if necessary.
18. Define an `Invoke` method in the main window declaration that calls the built-in `Invoke` method and additionally performs any actions required by the login window, such as entering a name and password.

    After following this procedure, your test frame might look like this:

```
window MainWin MyApp
   tag "My App"
   const wStartup = Login

   // the declarations for the MainWin should go here
   Invoke ()
      derived::Invoke ()
      Login.Name.SetText ("Your name")
      Login.Password.SetText ("password")
      Login.OK.Click ()

window DialogBox Login
   tag "Login"

   // the declarations for the Login window go here
   PushButton OK
      tag "OK"
```

    About the derived keyword and scope resolution operator

    Notice the statement `derived::Invoke ( )`. That statement uses the derived keyword followed by the scope resolution operator (`::`) to call the built-in Invoke method, before performing the operations needed to fill in and dismiss the login window.

Handling Login Windows in Non-Web Applications that Use the Open Agent

Although a non-Web application's main window is usually displayed first, it is also common for a login or security window to be displayed before the main window.

Use the `wStartup` constant and the `Invoke` method

To handle login windows, record a declaration for the login window, set the value of the `wStartup` constant, and write a new `Invoke` method for the main window that enters the appropriate information into the login window and dismisses it. This enables the `DefaultBaseState` routine to perform the actions necessary to get past the login window.

You do not need to use this procedure for splash screens, which disappear on their own.

1. Open the login window that precedes the application's main window.
2. Open the test frame.
3. Click **Record** > **Window Locators** to record a locator for the window.
4. Point to the title bar of the window and then press **Ctrl+Alt**.
   The locator is captured in the **Record Window Locators** dialog box.
5. Click **Paste to Editor** to paste the locator into the test frame.
6. In the **Record Window Locators** dialog box, click **Close**.
7. Close your application.
8. In your test frame file, find the stub of the declaration for the `wStartup` constant, located at the top of the declaration for the main window:

   ```
   // First window to appear when application is invoked
   // const wStartup = ?
   ```

9. Complete the declaration for the `wStartup` constant by:

   - Removing the comment characters, the two forward slash characters, at the beginning of the declaration.
   - Replacing the question mark with the identifier of the login window, as recorded in the window declaration for the login window.

10. Define an `Invoke` method in the main window declaration that calls the built-in `Invoke` method and additionally performs any actions required by the login window, such as entering a name and password.

    After following this procedure, your test frame might look like this:

    ```
    window MainWin MyApp
       locator "/MainWin[@caption='MyApp']"
       const wStartup = Login

       // the declarations for the MainWin should go here
       Invoke ()
          derived::Invoke ()
          Login.Name.SetText ("Your name")
          Login.Password.SetText ("password")
          Login.OK.Click ()

    window DialogBox Login
       locator "/DialogBox[@caption='Login']"

       // the declarations for the Login window go here
       PushButton OK
          locator "OK"
    ```

Note: Regarding the derived keyword and scope resolution operator. The statement `derived::Invoke ( )` uses the derived keyword followed by the scope resolution operator

(`::`) to call the built-in Invoke method, before performing the operations needed to fill in and dismiss the login window.

Handling Browser Pop-up Windows in Tests that Use the Classic Agent

Browser pop-up windows are recognized as instances of `Browser`.

When the popup window is active, it is seen as Browser and the original browser is seen as Browser 2. In order to make `DefaultBaseState`() close the pop-up window instead of the original browser, add the following line to the end of the test case:

```
Browser2.SetActive()
```

This is the standard way of ensuring that the pop-up becomes Browser2 and is closed by `DefaultBaseState`().

Specifying Windows to be Left Open for Tests that Use the Classic Agent

By default, the non-web recovery system closes all windows in your test application except the main window. To specify which windows, if any, need to be left open, such as a child window that is always open, use the `lwLeaveOpen` constant.

lwLeaveOpen constant

When you record and paste the declarations for your application's main window, the stub of a declaration for the `lwLeaveOpen` constant is automatically included, as shown in this example:

```
// The list of windows the recovery system is to leave open
// const lwLeaveOpen = {?}
```

To complete the declaration for the `lwLeaveOpen` constant:

1. Replace the question mark in the comment with the 4Test identifiers of the windows you want to be left open. Separate each identifier with a comma.
2. Remove the comment characters (the two forward slash characters) at the beginning of the declaration.

Example

he following 4Test code shows how to set the `lwLeaveOpen` constant so that the recovery system leaves open the window with the 4Test identifier `DocumentWindow` when it restores the base state.

```
const lwLeaveOpen = {DocumentWindow}
```

Specifying Windows to be Left Open for Tests that Use the Open Agent

By default, the non-Web recovery system closes all windows in your test application except the main window. To specify which windows, if any, need to be left open — such as a child window that is always open — use the `lwLeaveOpenWindows` or `lsLeaveOpenLocators` constant.

lwLeaveOpenWindows and lsLeaveOpenLocators constants

When you record and paste the declarations for your application's main window, the stub of a declaration for the `lwLeaveOpenWindows` constant is automatically included. Additionally, it is possible to specify

windows to leave open by using XPath locator strings. These can be specified with the variable `lsLeaveOpenLocators`, which must be a list of strings. The following example shows the `lwLeaveOpenWindows` and `lsLeaveOpenLocators` constants before they have been edited:

```
// The list of windows the recovery system is to leave open
// const lwLeaveOpenWindows = {?}
// const lsLeaveOpenLocators = {?}
```

To complete the declaration for these constants:

1. For `lwLeaveOpenWindows`, replace the question mark in the comment with the 4Test identifiers of the windows you want to be left open. Separate each identifier with a comma.
2. For `lsLeaveOpenLocators`, click **Record** > **Window Locators** and record the locators that you want to include.
3. Replace the question mark in the comment with the locator strings for the windows that you want to be left open. Separate each identifier with a comma.
4. Remove the comment characters (the two forward slash characters) at the beginning of the `lwLeaveOpenWindows` declaration.

   For example, the following code shows how to set the `lwLeaveOpenWindows` constant so that the recovery system leaves open the window with the identifier `DocumentWindow` when it restores the BaseState.

   ```
   const lwLeaveOpenWindows = {DocumentWindow}
   ```
5. Remove the comment characters (the two forward slash characters) at the beginning of the `lsLeaveOpenLocators` declaration.

   For example, the following code shows how to set the `lsLeaveOpenLocators` constant so that the recovery system leaves open the **About** dialog box when it restores the BaseState.

   ```
   lsLeaveOpenLocators = {"/MainWin[@caption='*Information*']", "//
   Dialog[@caption='About']"}
   ```

Specifying New Window Closing Procedures

When the recovery system cannot close a window using its normal procedure, you can reconfigure it in one of two ways:

- If the window can be closed by a button press, key press, or menu selection, specify the appropriate option either statically in the **Close** tab of the **Agent Options** dialog box or dynamically at runtime.
- Otherwise, record a `Close` method for the window.

This is only for classes derived from the `MoveableWin` class: `DialogBox`, `ChildWin`, and `MessageBox`. Specifying window closing procedures is not necessary for web pages, so this does not apply to `BrowserChild` objects/classes.

Specifying Buttons, Keys, and Menus that Close Windows

Specify statically

To specify statically the keys, menu items, and buttons that the non-Web recovery system should use to close all windows, choose **Options** > **Agent** and then click the **Close** tab.

The **Close** tab of the **Agent Options** dialog box contains a number of options, each of which takes a comma-delimited list of character string values.

Specify dynamically

As you set close options in the **Agent Options** dialog box, the informational text at the bottom of the dialog box shows the 4Test command you can use to specify the same option from within a script; add this 4Test command to a script if you need to change the option dynamically as a script is running.

Specify for individual objects

If you want to specify the keys, menu items, and buttons that the non-web recovery system should use to close an individual dialog box, define the appropriate variable in the window declaration for the dialog box:

- `lsCloseWindowButtons`
- `lsCloseConfirmButtons`
- `lsCloseDialogKeys`
- `lsCloseWindowMenus`

This is only for classes derived from the `MoveableWin` class: `DialogBox`, `ChildWin`, and `MessageBox`. Specifying window closing procedures is not necessary for web pages, so this does not apply to `BrowserChild` objects/classes.

Recording a Close Method for Tests that Use the Open Agent

To specify the keys, menu items, and buttons that the non-web recovery system uses to close an individual dialog box, record a `Close` method to define the appropriate variable in the window declaration for the dialog box.

1. Open your application.
2. Open the application's test frame file.
3. Choose **Record** > **Testcase**.
   SilkTest Classic displays the **Record Testcase** dialog box.
4. From the **Application state** list box, click `(None)`.
5. Click **Start Recording**.
   SilkTest Classic opens the **Recording** window, which indicates that you can begin recording the `Close` method.
6. When you have finished recording the `Close` method, click **Stop Recording** on the **Recording** window.
   SilkTest Classic redisplays the **Record Testcase** dialog box.
7. Click **Paste to Editor** and then copy and paste the script in the declaration for the dialog box in the test frame file.
8. Choose **File** > **Save** to save the test frame file.

You can also specify buttons, keys, and menus that close windows. This is only for classes derived from the `MoveableWin` class: `DialogBox`, `ChildWin`, and `MessageBox`. Specifying window closing procedures is not necessary for Web pages, so this does not apply to `BrowserChild` objects/classes.

Recording a Close Method for Tests that Use the Classic Agent

To specify the keys, menu items, and buttons that the non-Web recovery system uses to close an individual dialog box, record a `Close` method to define the appropriate variable in the window declaration for the dialog box.

1. Open your application.

2. Open the application's test frame file.
3. Place the insertion point on the window declaration for the dialog box.
4. Choose **Record** > **Method** .
5. From the **Method Name** list, select **Close** .
6. Click **Start Recording**.
   SilkTest Classic displays the **Record Status** dialog box, which indicates that you can begin recording the `Close` method. The **Status field** flashes the word `Recording`.
7. When you have finished recording the `Close` method, click **Done** on the **Record Status** dialog box. SilkTest Classic opens the **Record Method** dialog box. The **Method Code** field contains the 4Test code that you have recorded.
8. Click **OK** to close the **Record Method** dialog box and paste the new `Close` method in the declaration for the dialog box.

   You can also specify buttons, keys, and menus that close windows. This is only for classes derived from the `MoveableWin` class: `DialogBox`, `ChildWin`, and `MessageBox`. Specifying window closing procedures is not necessary for web pages, so this does not apply to `BrowserChild` objects/classes.

Creating a Test Plan

This section describes the test plan, which is a structured, usually hierarchical, document that describes the test requirements and contains the statements that implement the test requirements.

Overview of Test Plans

A test plan is a structured, usually hierarchical, document that describes the test requirements and contains the statements, 4Test scripts and test cases, that implement the test requirements. A test plan is displayed in an easy-to-read outline format, which lists the test requirements in high-level prose descriptions. The structure can be flat or many levels deep.

Indentation and color indicate the level of detail and various test plan elements. Large test plans can be divided into a master plan and one or more sub-plans. A test plan file has a .pln extension, such as `find.pln`.

When you structure your test plan as a hierarchical outline you:

- Assist the test plan author in developing thoughts about the test problem by promoting and supporting a top-down approach to test planning.
- Yield a comprehensive inventory of test requirements, from the most general, through finer and finer levels of detail, to the most specific.
- Allow the statements that actually implement the tests to be shared by group descriptions or used by just a single test description.
- Provide reviewers with a framework for evaluating the thoroughness of the plan and for following the logic of the test plan author.
- If you are using the test plan editor, the first step in creating automated tests is to create a test plan. If you are not using the test plan editor, the first step is creating a test frame.

Structure of a Test Plan

A test plan is made up of the following elements, each of which is identified by color and indentation on the test plan.

Element	Description	Color
Comment	Provide documentation throughout the test plan; preceded by `//`.	Green
Group Description	High level line in the test requirements outline that describes a group of tests.	Black
Test Description	Lowest level line describing a single test case; is a statement of the functionality to be tested by the associated test case.	Blue
Test Plan Statement	Used to provide script name, test case name, test data, or `include` statement.	Red when a sub plan is not expanded. Magenta statement when sub-plan is expanded

A statement placed at the group description level applies to all the test descriptions contained by the group. Conversely, a statement placed at the test description level applies only to that test description. Levels in the test plan are represented by indentation.

Because there are many ways to organize information, you can structure a test plan using as few or as many levels of detail as you feel are necessary. For example, you can use a list structure, which is a list of test descriptions with no group description, or a hierarchical structure, which is a group description and test description. The goal when writing test plans is to create a top-down outline that describes all of the test requirements, from the most general to the most specific.

Overview of Test Plan Templates

Because a test plan is initially empty, you may want to insert a template, which is a hierarchical outline you can use as a guide when you create a new test plan.

The template contains placeholders for each GUI object in your application. Although you may not want to structure the test plan in a way which mirrors the hierarchy of your application's GUI, this can be a good starting point if you are new to creating test plans.

In order to be able to insert a template, you must first record a test frame, which contains declarations for each of the GUI objects in your application.

Example Outline for Word Search Feature

Because a test plan is made up of a large amount of information, a structured, hierarchical outline provides an ideal model for organizing and developing the details of the plan. You can structure an outline using as few or as many levels of detail as you feel necessary.

The following is a series of sample outlines, ranging from a simple *list structure* to a more specific *hierarchical structure*. For completeness, each of the plans also shows the script and test case statements that link the descriptions to the 4Test scripts and test cases that implement the test requirements.

For example, consider the **Find** dialog box from the Text Editor application, which allows a user to search in a document. A user enters the characters to search for in the **Find What** text box, checks the **Case sensitive** check box to consider case, and clicks either the **Up** or **Down** radio button to indicate the direction of the search.

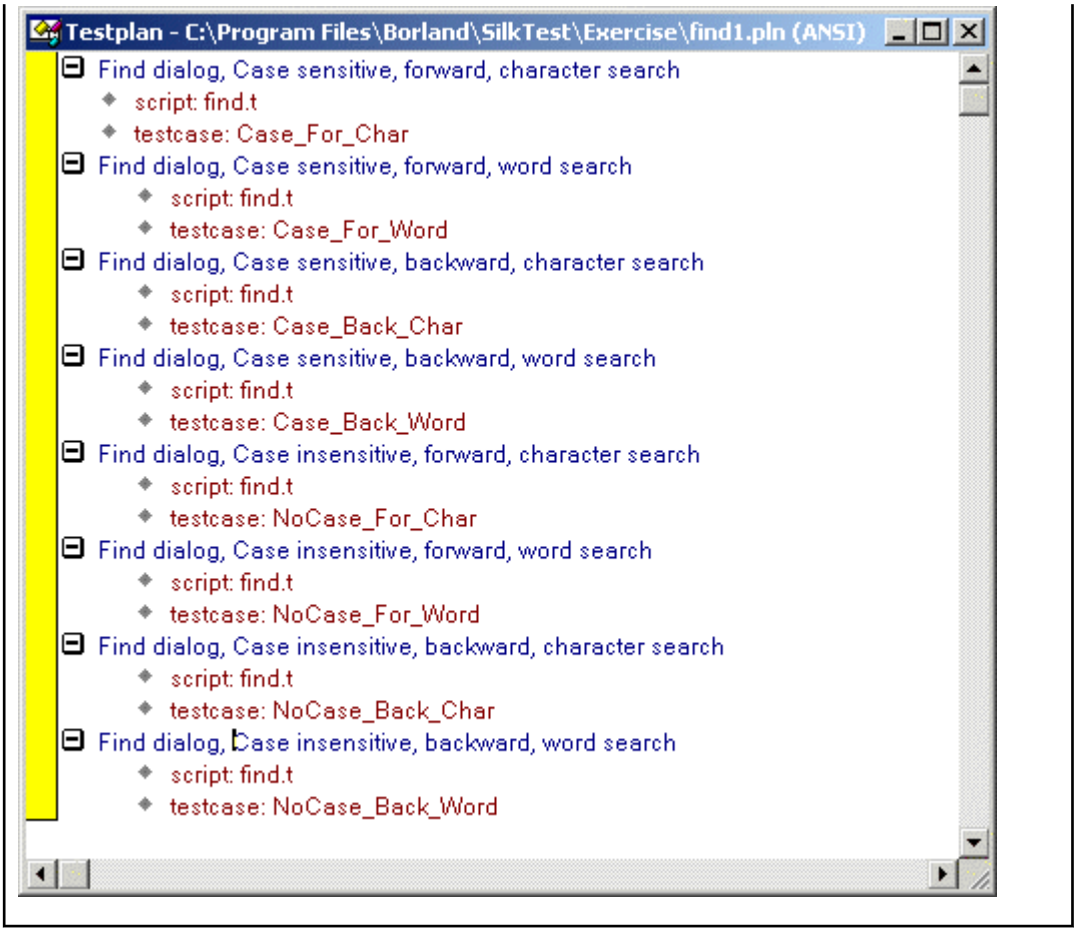
List Structure

At its simplest, an outline is a hierarchy with just a single level of detail. In other words, it is a list of test descriptions, with no group descriptions.

Using the list structure, each test is fully described by a single line, which is followed by the script and test case that implement the test. You may find this style of plan useful in the beginning stages of test plan design, when you are brainstorming the list of test requirements, without regard for the way in which the test requirements are related. It is also useful if you are creating an ad hoc test plan that runs a set of unrelated 4Test scripts and test cases.

Example for List Structure

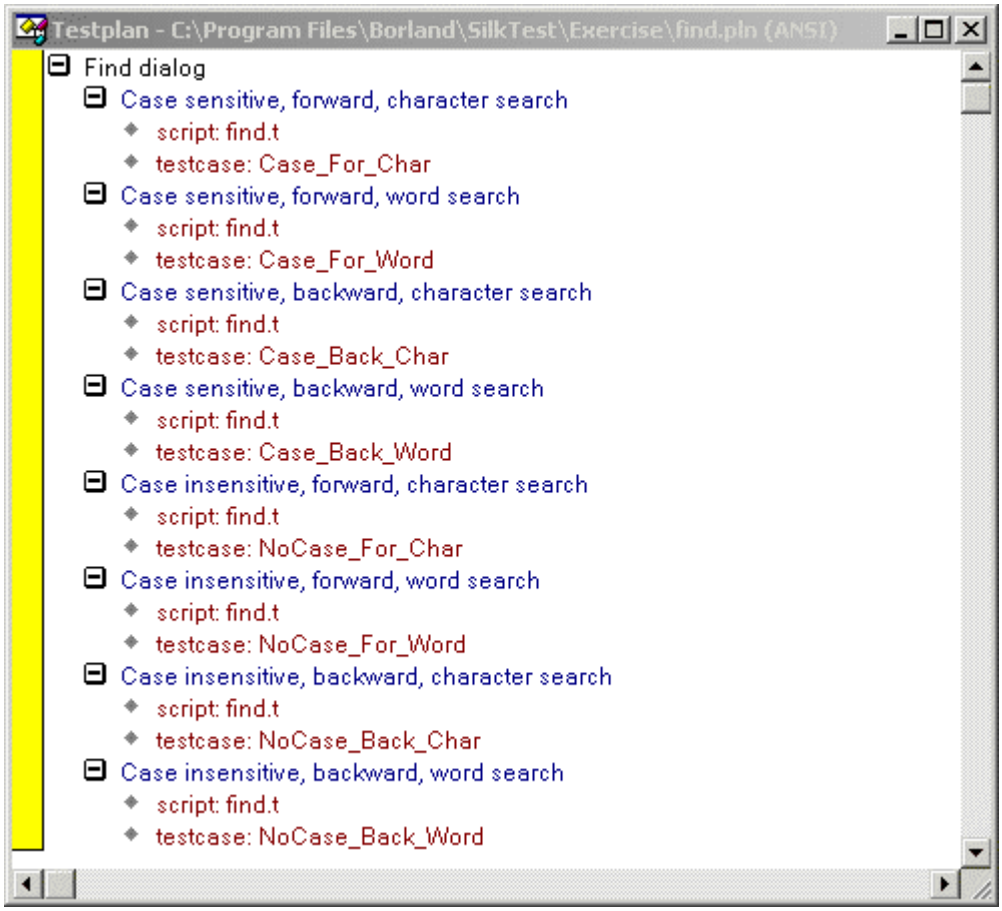
For example:



Hierarchical Structure

The following test plan has a single level of group description, preceding the level that contains each of the test descriptions. The group description indicates that all the tests are for the **Find** dialog box.

As the figure shows, the test plan editor indicates levels in the outline with indentation. Each successive level is indented one level to the right. The minus icons indicate that each of the levels is fully expanded. By clicking on the minus icon at any level, you collapse the branch below that level. When working with large test plans, collapsing and expanding test plan detail makes it easy to see as much or as little of the test plan as you need. You could continue this test plan by adding a second level of group description, indicating whether or not the tests in the group are case sensitive, and even more detail by adding a third level of group descriptions which indicate whether the tests in the group search in the forward or backward direction.



Converting a Results File to a Test Plan

Using SilkTest Classic, you can convert a results file into a test plan. This is useful when converting suite-based tests into test plans.

1. Open a results file that was generated by SilkTest Classic, not one generated by the test plan editor from a test plan.
2. Click **Results** > **Convert to Plan**.
3. Select the results file you want to convert, which is typically the most recent, and click **OK**. The test plan editor converts the results file to a test plan.

   When creating a test plan from a results file generated for a script, the test plan editor uses the # symbol so that when this test plan is run, the `testdata` statement doubles as description. Since the results file was for a script, not a test plan, it does not contain any group or test case descriptions. The # symbol can be used with any test plan editor statement so that the statement will double as description.

Working with Test Plans

This section describes how you can work with test plans.

Creating a New Test Plan

1. Click **File** > **New**.
2. Click **Testplan** and click **OK**.

An empty test plan window opens.

3. Create your test plan and then click **File** > **Save**.
4. Specify the name and location in which to save the file, and then click **OK**.
5. If you are working within a project, SilkTest Classic prompts you to add the file to the project. Click **Yes** if you want to add the file to the open project, or **No** if you do not want to add this file to the project.

Before you can begin testing, you must enable extensions for applications you plan to test on both the target machine and the host machine.

Indent and Change Levels in an Outline

You can use menu, keyboard, or toolbar commands to enter or change group and test descriptions as you are typing them. The following table summarizes the commands:

Action	Menu Item	Key	Tool
Indent one level	Outline/Move Right	**ALT + forward arrow**	
Outdent one level	Outline/Move Left	**ALT + back arrow**	
Swap with line above	Outline/Transpose Up	**ALT + up arrow**	None
Swap with line below	Outline/Transpose Down	**ALT + down arrow**	None

Each command acts on the current line or currently selected lines.

SilkTest Classic ignores comments when compiling, with the exception of functions and test cases. Comments within functions and test cases must be within the scope of the function/test case. If a comment is outdented beyond the scope of the function/test case, the compiler assumes that the function/test case has ended. As long as comments do not violate the function/test case scope, they can be placed anywhere on a line.

Note: Comments beyond the scope can also impact expand/collapse functionality and may prevent a function/test case from being fully expanded/collapsed. We recommend that you keep comments within scope.

Add Comments to a Test Plan that Display in the Results

You can add comments to your test plans that will display in the results when you run your tests. Comments are a handy way to annotate your tests to make it easier to interpret your results.

To add a comment, include the following statement in your plan:

```
comment: Your comment text
```

For example, running the following piece of a test plan:

```
Find dialog
  Get the default button
    comment: This test should return Find.FindNext
    script: find.t
    testcase: GetButton
```

produces the following in the results file:

```
Find dialog
  Get the default button
    Find.FindNext
    comment: This test should return Find.FindNext
```

You can also preface lines in all 4Test files with // to indicate a single-line comment. Such comments do not display in test plan results.

Documenting Manual Tests in the Test Plan

Your QA department might do some of its testing manually. You can document the manual testing in the test plan. In this way, the planning, organization, and reporting of all your testing can be centralized in one place. You can describe the state of each of your manual tests. This information is used in reports.

To indicate that a test description in the test plan is implemented with a manual test, use the value `manual` in the `testcase` statement, as in:

```
testcase: manual
```

By default, whenever you generate a report, it includes information on the tests run for that results file, plus the current results of any manual tests specified in the test plan. If the manual test results are subsequently updated, the next time you generate the report, it incorporates the latest manual results. However, this might not be what you want. If you want the report to use a snapshot of manual results, not the most recent manual results, merge the results of manual tests into the results file.

Describing the State of a Manual Test

1. Open a test plan containing manual tests.
2. Click **Testplan** > **Run Manual Tests**.
3. Select a manual test from the **Update Manual Tests** dialog box and document it. The **Update Manual Tests** dialog box lists all manual tests in the current test plan.

Mark the test complete	Click the **Complete** option button. `Complete` means that a test has been defined. A manual test marked here as `Complete` will be tabulated as complete in Completion reports.
Indicate whether the test passed or failed	1. Click the **Has been run** option button. 2. Select **Passed** or **Failed**. 3. Specify when the test was run and optionally, specify the machine. To specify when the test was run, use this syntax: `YYYY-MM-DD HH:MM:SS` Hours, minutes, and seconds are optional. For example, enter `2006-01-10` to indicate the test was run Oct 1, 2006. Manual tests marked as `Passed` or `Failed` will be tabulated as such in Pass/Fail reports, as long as you have also specified the time they were run. A test marked `Has been run` is also considered complete in Completion reports.
Add any comments you want about the test	Fill in the **Comments** text box.

Inserting a Template

1. Click **Testplan** > **Insert Template**.
   The **Insert Testplan Template** dialog box, which lists all the GUI objects declared in your test frame, opens.
2. Select each of the GUI objects that are related to the application features you want to test.
   Because this is a multi-select list box, the objects do not have to be contiguous.
   For each selected object, SilkTest Classic inserts two lines of descriptive text into the test plan.

For example, the test plan editor would create the following template for the **Find** dialog box of the Text Editor application:

```
Tests for DialogBox Find
Tests for StaticText FindWhatText
(Insert tests here)
Tests for TextField FindWhat
(Insert tests here)
Tests for CheckBox CaseSensitive
(Insert tests here)
Tests for StaticText DirectionText
(Insert tests here)
Tests for PushButton FindNext
(Insert tests here)
Tests for PushButton Cancel
(Insert tests here)
Tests for RadioList Direction
(Insert tests here)
```

Changing Colors in a Test Plan

You can customize your test plan so that different test plan components display in unique colors.

To change the default colors:

1. Click **Options** > **Editor Colors**.
2. On the **Editor Colors** dialog box, select the outline editor item you want to change in the **Editor Item** list box at the left of the dialog box.
3. Apply a color to the item by selecting a pushbutton from the list of predefined colors or create a new color to apply by selecting the red, green, and blue values that compose the color.

Default color	Component	Description
Blue	Test description	Lowest level of the hierarchical test plan outline that describes a single testcase.
Red	Test plan statement	Link scripts, test cases, test data, closed sub-plans, or an include file (such as a test frame) to the test plan.
Magenta	Include statement when sub-plan is open	Sub-plans to be included in a master plan.
Green	Comment	Additional user information that is incidental to the outline; preceded by double slashes (//); provides documentation throughout the test plan.
Black	Other line (group description)	Higher level lines of the hierarchical test plan outline that describe a group of tests; may be several levels in depth.

Linking the Test Plan to Scripts and Test Cases

After you create your test plan, you can associate the appropriate 4Test scripts and test cases that implement your test plan. You create this association by inserting `script` and `testcase` statements in the appropriate locations in the test plan.

There are three ways to link a script or test case to a test plan:

- Linking a description to a script or test case using the **Testplan Detail** dialog box if you want to automate the process of linking scripts and test cases to the test plan.
- Linking to a test plan manually.
- Linking scripts and test cases to a test plan: the test plan editor automatically inserts the `script` and `testcase` statements into the plan once the recording is finished, linking the plan to the 4Test code.

You can insert a `script` and `testcase` statement for each test description, although placing a statement at the group level when possible eliminates redundancy in the test plan. For example, since it is usually

good practice to place all the test cases for a given application feature into a single script file, you can reduce the redundancy in the test plan by specifying the `script` statement at the group level that describes that feature.

You can also insert a `testcase` statement at the group level, although doing so is only appropriate when the test case is data driven, meaning that it receives test data from the plan. Otherwise the same test case would be called several times with no difference in outcome.

Working with Large Test Plans

This section describes how you can divide large test plans into a set of master plans and sub-plans.

Overview of Working with Large Test Plans

For large or complicated applications, the test plan can become quite large. This raises the following issues:

- How to keep track of where you are in the plan and what is in scope at that level.
- How to determine which portions of the plan have been implemented.
- How to allow several staff members to work on the plan at the same time.

To determine context and scope, you can use the **Testplan Detail** dialog box. To determine which portions of the plan have been implemented, you can produce a **Completion** report. To allow multiple users to work on the same plan, you can structure your test plan as a master plan with one or more sub-plans.

Determining Where Values are Defined in a Large Test Plan

1. Place the insertion point at the relevant point in the test plan and choose **Testplan** > **Detail** to open the **Testplan Detail** dialog box.
2. To see just the set of symbols, attributes, and statements that are defined on a particular level, click the level in the list box at the top of the **Testplan Detail** dialog box.
3. Once you find the level at which a symbol, attribute, or statement was defined, you can change the value at that level, causing the inherited value at the lower levels to change also.

Dividing a Test Plan into a Master Plan and Sub-Plans

If several engineers in your QA department will be working on a test plan, it makes sense to break up the plan into a master plan and sub-plans. This approach allows multi-user access, while at the same time maintaining a single point of control for the entire project.

The master plan contains only the top few levels of group descriptions, and the sub-plans contain the remaining levels of group descriptions and test descriptions. Statements, attributes, symbols, and test data defined in the master plan are accessible within each of the sub-plans.

Sub-plans are specified with an `include` statement. To expand the sub-plan files so that they are visible within the master plan, double-click in the left margin next to the `include` statement. Once a sub-plan is expanded inline, the sub-plan statement changes from red (the default color for statements) to magenta, indicating that the line is now read-only and that the sub-plan is expanded inline. At the end of the expanded sub-plan is the `<eof>` marker, which indicates the end of the sub-plan file.

Creating a Sub-Plan

You create a sub-plan in the same way you create any test plan: by opening a new test plan file and entering the group descriptions, test descriptions, and the test plan editor statements that comprise the sub-plan, either manually or using the **Testplan Detail** dialog.

Copying a Sub-Plan

When you copy and paste the include statement and the contents of an open include file, note that only the include statement will be pasted.

To view the contents of the sub-plan, open the pasted include file by clicking **Include** > **Open** or double-click the margin to the left of the include statement.

Opening a Sub-Plan

Open the sub-plan from within the master plan. To do this, you can either:

- double-click the margin to the left of the include statement or
- highlight the include statement and choose **Include** > **Open**. (Compiling a script also automatically opens all sub-plans.)

If a sub-plan does not inherit anything (that is, statements, attributes, symbols, or data) from the master plan, you can open the sub-plan directly from the **File** > **Open** dialog box.

Connecting a Sub-Plan with a Master Plan

To connect the master plan to a sub-plan file, you enter an `include` statement in the master plan at the point where the sub-plan logically fits. The `include` statement cannot be entered through the **Testplan Detail** dialog box; you must enter it manually.

The `include` statement uses this syntax:

```
include:  myinclude.pln
```

where `myinclude` is the name of the test plan file that contains the sub-plan.

If you enter the `include` statement correctly, it displays in red, the default color used for the test plan editor statements. Otherwise, the statement displays in blue or black, indicating a syntax error (the compiler is interpreting the line as a description, not a statement).

Refreshing a Local Copy of a Sub-Plan

When another user modifies a sub-plan, those changes are not automatically reflected in your read-only copy of the sub-plan. Once the other user has released the lock on the sub-plan, there are two ways to refresh your copy:

1. Close and then reopen the sub-plan.
2. Acquire a lock for the sub-plan.

Sharing a Test Plan Initialization File

All QA engineers working on a test plan that is broken up into a master plan and sub-plans must use the same test plan initialization file.

To share a test plan initialization file:

1. Click **Options** > **General**.

2. On the **General Options** dialog box, specify the same file name in the **Data File for Attributes and Queries** text box.

Saving Changes

When you finish editing, choose **Include** > **Save** to save the changes to the sub-plan.

Include > **Save** saves changes to the current sub-plan while **File** > **Save** saves all open master plans and sub-plans.

Overview of Locks

When first opened, a master plan and its related sub-plans are read-only. This allows many users to open, read, run, and generate reports on the plan. When you need to edit the master plan or a sub-plan, you must first acquire a lock, which prevents others from making changes that conflict with your changes.

Acquiring and Releasing a Lock

Acquire a lock Place the cursor in or highlight one or more sub-plans and then choose **Include** > **Acquire Lock**.

The bar in the left margin of the test plan changes from gray to yellow.

Release a lock Select **Include** > **Release Lock**.

The margin bar changes from yellow to gray.

Generating a Test Plan Completion Report

To measure your QA department's progress in implementing a large test plan, you can generate a completion report. The completion report considers a test complete if the test description is linked to a test case with two exceptions:

- If the test case statement invokes a data-driven test case and a symbol being passed to the data-driven test case is assigned the value ? (undefined), the test is considered incomplete.
- If the test case is manual and marked as Incomplete in the **Update Manual Tests** dialog box, the test is considered incomplete. A manual test case is indicated with the `testcase:manual` syntax.

To generate a test plan completion report:

1. Open the test plan on which you want to report.
2. Click **Testplan** > **Completion Report** to display the **Testplan Completion Report** dialog box.
3. In the **Report Scope** group box, indicate whether the report is for the entire plan or only for those tests that are marked.
4. To subtotal the report by a given attribute, select an attribute from the **Subtotal by Attribute** text box.
5. Click **Generate**.

   The test plan editor generates the report and displays it in the lower half of the dialog box. If the test plan is structured as a master plan with associated sub-plans, the test plan editor opens any closed sub-plans before generating the report.

   You can:

   - Print the report.
   - Export the report to a comma-delimited ASCII file. You can then bring the report into a spreadsheet application that accepts comma-delimited data.
   - Chart (graph) the report, just as you can chart a Pass/Fail report.

Adding Data to a Test Plan

This section describes how you can add data to a test plan.

Specifying Unique and Shared Data

If a data value is unique to a single test description	You should place it in the plan at the same level as the test description, using the `testdata` statement. You can add the `testdata` statement using the **Testplan Detail** dialog box or type the `testdata` statement directly into the test plan.
If data is common to several tests	You can factor out the data that is common to a group of tests and define it at a level in the test plan where it can be shared by the group. To do this, you define symbols and assign them values. Using symbols results in less redundant data, and therefore, less maintenance.

Adding Comments in the Test Plan Editor

Use two forward slash characters to indicate that a line in a test plan is a comment. For example:

```
// This is a comment
```

Comments preceded by `//` do not display in the results file. You can also specify comments using the comment statement; these comments will display in the results files.

Testplan Editor Statements

You use the test plan editor keywords to construct statements, using this syntax:

```
keyword : value
```

`keyword`: One of the test plan editor keywords.

`value`: A comment, script, test case, include file, attribute name, or data value.

For example, this statement associates the script `myscript.t` with the plan:

```
script : myscript.t
```

Spaces before and after the colon are optional.

The # Operator in the Testplan Editor

When a # character precedes a statement, the statement will double as a test description in the test plan. This helps eliminate possible redundancies in the test plan. For example, the following test description and script statement:

```
Script is test.t
        script:test.t
```

can be reduced to one line in the test plan:

```
#script: test.t
```

The test plan editor considers this line an executable statement as well as a description. Any statements that follow this "description" in the test plan and that trigger test execution must be indented.

Using the Testplan Detail Dialog Box to Enter the testdata Statement

1. Place the insertion point at the end of the test description.
   If a `testdata` statement is not associated with a test description, the compiler generates an error.
2. Click **Testplan** > **Detail**.
   To provide context, the multi-line list box at the top of the **Testplan Detail** dialog box displays the line in the test plan that the cursor was on when the dialog box was invoked, indicated by the black arrow icon. If the test case and script associated with the current test description are inherited from a higher level in the test plan, they are shown in blue; otherwise, they are shown in black.
3. Enter the data in the **Test Data** text box, separating each data element with a comma.

   Remember, if the test case expects a record, you need to enclose the list of data with the list constructor operator (the curly braces); otherwise, SilkTest Classic interprets the data as individual variables, not a record, and will generate a data type mismatch compiler error.
4. Click **OK**.
   SilkTest Classic closes the **Testplan Detail** dialog box and enters the testdata statement and data values in the plan.

Entering the testdata Statement Manually

1. Open up a new line after the test description and indent the line one level.
2. Enter the testdata statement as follows.

   - If the test case expects one or more variables, use this syntax: `testdata: data [,data]`, where data is any valid 4Test expression.
   - A record, use the same syntax as above, but open and close the list of record fields with curly braces: `testdata: {data [,data]}`, where data is any valid 4Test expression.

   Be sure to follow the testdata keyword with a colon. If you enter the keyword correctly, the statement displays in dark red, the default color. Otherwise, the statement displays in either blue or black, indicating the compiler is interpreting the line as a description.

Linking Test Plans

This section describes how SilkTest Classic handles linking from a test plan to a script or test case.

Linking a Description to a Script or Test Case using the Testplan Detail Dialog Box

1. Place the insertion cursor on either a test description or a group description.
2. Click **Testplan** > **Detail**.
   The test plan editor invokes the **Testplan Detail** dialog box, with the **Test Execution** tab showing. The multi-line list box at the top of the dialog box displays the line in the test plan that the cursor was on when the dialog box was invoked, as well as its ancestor lines. The black arrow icon indicates the current line. The current line appears in black and white, and the preceding lines display in blue.
3. If you:

   - know the names of the script and test case, enter them in the **Script** and **Testcase** fields, respectively.

- are unsure of the script name, click the **Scripts** button to the right of the **Script** field to browse for the script file.

4. On the **Testplan Detail - Script** dialog box, navigate to the appropriate directory and select a script name by double-clicking or by selecting and then clicking **OK**.
   SilkTest Classic closes the **Testplan Detail - Script** dialog box and enters the script name in the **Script** field.
5. Click the **Testcases** button to the right of the **Testcase** field, to browse for the test case name.

   The **Testplan Detail – Testcase** dialog box shows the names of the test cases that are contained in the selected script. Test cases are listed alphabetically, not in the order in which they occur in the script.
6. Select a test case from the list and click **OK**.
7. Click **OK**.
   The script and test case statements are entered in the plan.

If you feel comfortable with the syntax of the test plan editor statements and know the locations of the appropriate script and test case, you can enter the script and test case statements manually.

Linking a Test Plan to a Data-Driven Test Case

To link a group of test descriptions in the plan with a data-driven test case, add the test case declaration to the group description level. There are three ways to do this:

- Linking a test case or script to a test plan using the **Testplan Detail** dialog box to automate the process.
- Link to a test plan manually.
- Record the test case from within the test plan.

Linking to a Test Plan Manually

If you feel comfortable with the syntax of the test plan editor statements and know the locations of the appropriate script and test case, you can enter the `script` and `testcase` statements manually.

1. Place the insertion cursor at the end of a test or group description and press **Enter** to create a new line.
2. Indent the new line one level.
3. Enter the script and/or test case statements using the following syntax:

   ```
   script:
   scriptfilename.t testcase:
   testcasename
   ```

   Where script and testcase are keywords followed by a colon, `scriptfilename.t` is the name of the script file, and `testcasename` is the name of the test case.

   If you enter a statement correctly, it displays in dark red, the default color used for statements. If not, it will either display in blue, indicating the line is being interpreted as a test description, or black, indicating it is being interpreted as a group description.

Linking a Test Case or Script to a Test Plan using the Testplan Detail Dialog Box

The **Testplan Detail** dialog box automates the process of linking to scripts and test cases. It lets you browse directories and select script and test case names, and it enters the correct the test plan editor syntax into the plan for you.

Linking the Test Plan to Scripts and Test Cases

After you create your test plan, you can associate the appropriate 4Test scripts and test cases that implement your test plan. You create this association by inserting `script` and `testcase` statements in the appropriate locations in the test plan.

There are three ways to link a script or test case to a test plan:

- Linking a description to a script or test case using the **Testplan Detail** dialog box if you want to automate the process of linking scripts and test cases to the test plan.
- Linking to a test plan manually.
- Linking scripts and test cases to a test plan: the test plan editor automatically inserts the `script` and `testcase` statements into the plan once the recording is finished, linking the plan to the 4Test code.

You can insert a `script` and `testcase` statement for each test description, although placing a statement at the group level when possible eliminates redundancy in the test plan. For example, since it is usually good practice to place all the test cases for a given application feature into a single script file, you can reduce the redundancy in the test plan by specifying the `script` statement at the group level that describes that feature.

You can also insert a `testcase` statement at the group level, although doing so is only appropriate when the test case is data driven, meaning that it receives test data from the plan. Otherwise the same test case would be called several times with no difference in outcome.

Example of Linking a Test Plan to a Test Case

For example, consider the data driven test case FindTest, which takes a record of type `SEARCHINFO` as a parameter:

```
type SEARCHINFO is record
   STRING  sText      // Text to type in document window
   STRING  sPos       // Starting position of search
   STRING  sPattern   // String to look for
   BOOLEAN bCase      // Case-sensitive or not
   STRING  sDirection // Direction of search
   STRING  sExpected  // The expected match

testcase FindTest (SEARCHINFO Data)
   TextEditor.File.New.Pick ()
   DocumentWindow.Document.TypeKeys (Data.sText + Data.sPos)
   TextEditor.Search.Find.Pick ()
   Find.FindWhat.SetText (Data.sPattern)
   Find.CaseSensitive.SetState (Data.bCase)
   Find.Direction.Select (Data.sDirection)
   Find.FindNext.Click ()
   Find.Cancel.Click ()
   DocumentWindow.Document.VerifySelText ({Data.sExpected})
   TextEditor.File.Close.Pick ()
   MessageBox.No.Click ()
```

The following test plan is associated with the `FindTest testcase` (the `testcase` statement is highlighted for emphasis). The statement occurs at the **Find** dialog group description level, so that each of the test descriptions in the group can call the test case, passing it a unique set of data:

```
Testplan FindTest.pln

Find dialog
script: findtest.t
testcase: FindTest
. . . .
```

Categorizing and Marking Test Plans

This section describes how you can work with selected tests in a test plan.

Marking a Test Plan

Marks are temporary denotations that allow you to work with selected tests in a test plan. For example, you might want to run only those tests that exercise a particular area of the application or to report on only the tests that were assigned to a particular QA engineer. To work with selected tests rather than the entire test plan, you denote or **mark** those tests in the test plan.

Marks can be removed at any time, and last only as long as the current work session. You can recognize a marked test case by the black stripe in the margin.

You can mark test cases by:

Choice	Select the individual test description, group description, or entire plan that you want to mark, and then choosing the appropriate marking command on the **Testplan** menu.
Query	You can also mark a test plan according to a certain set of characteristics it possesses. This is called marking by query. You build a query based on one or more specific test characteristics; its script file, data, symbols, or attributes, and then mark those tests that match the criteria set up in the query. For example, you might want to mark all tests that live in the `find.t` script and that were created by the developer named Peter. If you name and save the query, you can reapply it in subsequent work sessions without having to rebuild the query or manually remark the tests that you're interested in working with.
Test failure	After running a test plan, the generated results file might indicate test failures. You can mark these failures in the plan by selecting **Results** > **Mark Failures in Plan**. You then might fix the errors and re-run the failed tests.

How the Marking Commands Interact

When you apply a mark using the **Mark** command, the new mark is added to existing marks.

When you mark tests through the query marking commands, the test plan editor by default clears all existing marks before running the query. **Mark by Named Query** supports sophisticated query combinations, and it would not make sense to retain previous marks. However, **Mark by Query**, which allows one-time-only queries, lets you override the default behavior and retain existing marks.

To retain existing marks, uncheck the **Unmark All Before Query** check box in the **Mark by Query** dialog box.

Marking One or More Tests

To mark:

A single test	Place the cursor on the test description and click **Testplan** > **Mark**.
A group of related tests	Place the cursor on the group description and click **Testplan** > **Mark**. The test plan editor marks the group description, its associated statements, and all test descriptions and statements subordinate to the group description.
Two or more adjacent tests and their subordinate tests	Select the test description of the adjacent tests and click **Testplan** > **Mark**. The test plan editor marks the test descriptions and statements of each selected test and any subordinate tests.

Printing Marked Tests

1. Click **File** > **Print**.
2. In the **Print** dialog box, make sure the **Print Marked Only** check box is checked, as well as any other options you want.
3. Click **OK**.

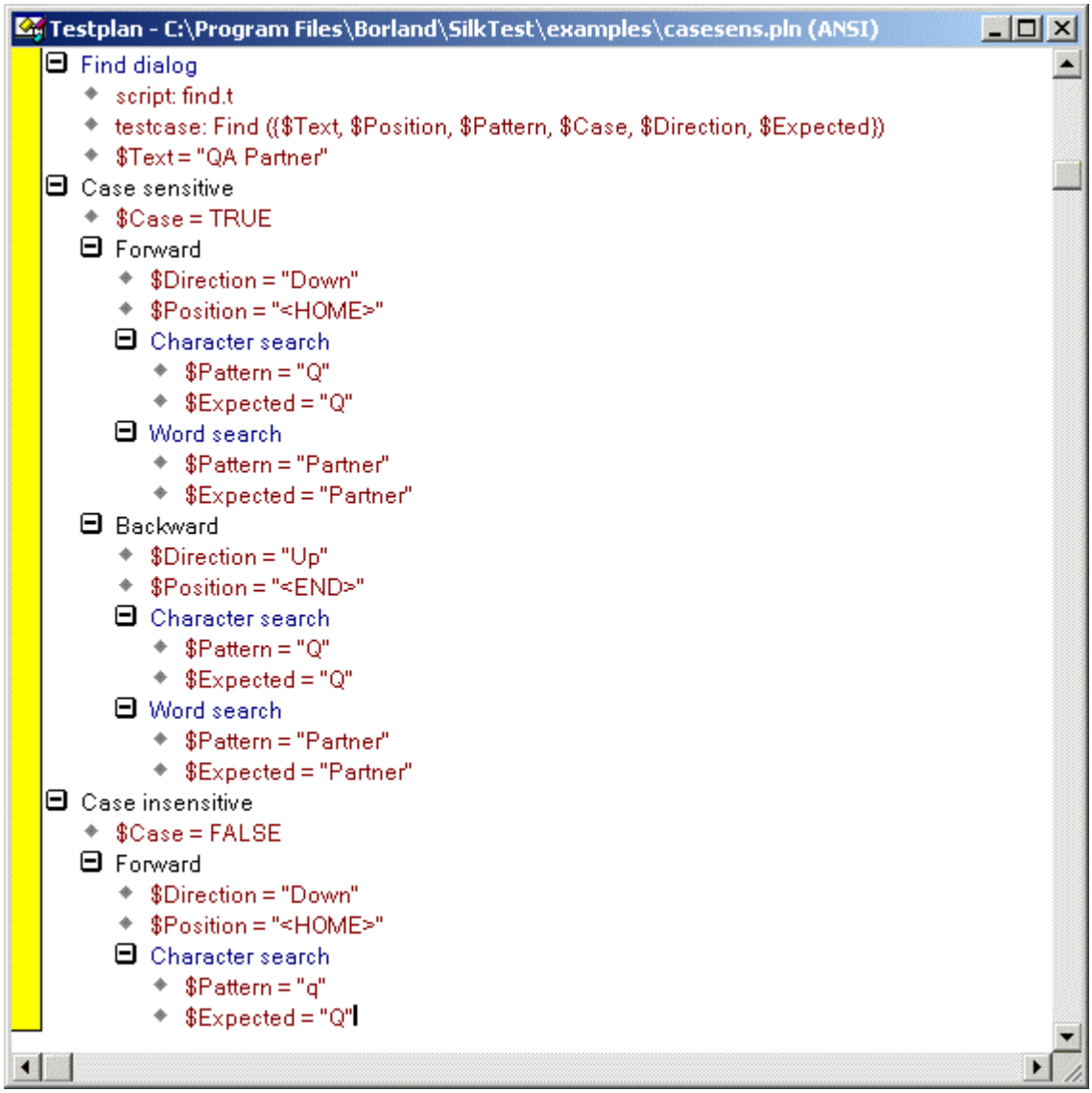
Using Symbols

This section describes symbols, which represent pieces of data in a data driven test case.

Overview of Symbols

A symbol represents a piece of data in a data driven test case. It is like a 4Test identifier, except that its name begins with the $ character. The value of a symbol can be assigned locally or inherited. Locally assigned symbols display in black and symbols that inherit their value display in blue in the **Testplan Detail** dialog box.

For example, consider the following test plan:



The test plan in the figure uses six symbols:

- `$Text` is the text to enter in the document window.
- `$Position` is the position of the insertion point in the document window.
- `$Pattern` is the pattern to search for in the document window.
- `$Case` is the state of the **Case Sensitive** check box.
- `$Direction` is the direction of the search.
- `$Expected` is the expected match.

The symbols are named in the parameter list to the `FindTest testcase`, within the parentheses after the test case name.

```
testcase: FindTest ({ $Text, $Position, $Pattern, $Case, $Direction,
$Expected })
```

- The symbols are only named in the parameter list; they are not assigned values. The values are assigned at either the group or test description level, depending on whether the values are shared by several tests or are unique to a single test. If a symbol is defined at a level in the plan where it can be shared by a group of tests, each test can assign its own local value to the symbol, overriding whatever value it had at the higher level. You can tell whether a symbol is locally assigned by using the **Testplan Detail** dialog box: Locally assigned symbols display in black. Symbols that inherit their values display in blue.

For example, in the preceding figure, each test description assigns its own unique values to the $Pattern and the $Expected symbols. The remaining four symbols are assigned values at a group description level:

- The `$Text` symbol is assigned its value at the Find dialog group description level, because all eight tests of the Find dialog enter the text SilkTest into the document window of the Text Editor application.
- The `$Case` symbol is assigned the value TRUE at the Case sensitive group description level and the value FALSE at the Case insensitive group description level.
- The `$Direction` symbol is assigned the value Down at the Forward group description level, and the value Up at the Backward group description level.
- The `$Position` symbol is assigned the value <HOME> at the Forward group description level, and the value <END> at the Backward group description level.

Because the data that is common is factored out and defined at a higher level, it is easy to see exactly what is unique to each test.

Symbol Definition Statements in the Test Plan Editor

Use symbols to define data that is shared by a group of tests in the plan. Symbol definitions follow these syntax conventions:

- The symbol name can be any valid 4Test identifier name, but must begin with the $ character.
- The symbol value can be any text. When the test plan editor encounters the symbol, it expands it (in the same sense that another language expands macros). For example, the following test plan editor statement defines a symbol named Color and assigns it the `STRING` value "Red":

  ```
  $Color = "Red"
  ```
- To use a $ in a symbol value, precede it with another $. Otherwise, the compiler will interpret everything after the $ as another symbol. For example, this statement defines a symbol with the value Some $String: `$MySymbol = "Some$$String "`
- To assign a null value to a symbol, do not specify a value after the equals sign. For example: `$MyNullSymbol =`
- To indicate that a test is incomplete when generating a test plan completion report, assign the symbol the ? character. For example: `$MySymbol = ?`

If a symbol is listed in the argument list of a test case, but is not assigned a value before the test case is actually called, the test plan editor generates a runtime error that indicates that the symbol is undefined. To avoid this error, assign the symbol a value or a ? if the data is not yet finalized.

Defining Symbols in the Testplan Detail Dialog box

Place the insertion cursor in the plan where you need to assign a value to a symbol.

1. Click **Testplan** > **Detail**.
2. Select the **Symbols** tab on the **Testplan Detail** dialog box, and enter the symbol definition in the text box to the left of the **Add** button.

   You do not need to enter the $ character; the test plan editor takes care of this for you when it inserts the definitions into the test plan.
3. Click **Add**.
   SilkTest Classic adds the symbol to the list box above the **Add text** text box.
4. Define additional symbols in the same manner, and then click **OK** when finished.

   SilkTest Classic closes the **Testplan Detail** dialog box and enters the symbol definitions, including the $ character, into the plan. If a symbol is defined at a level in the plan where it can be shared by a group of tests, each test can assign its own local value to the symbol, overriding whatever value it had at the higher level. You can tell whether a symbol is locally assigned by using the **Testplan Detail** dialog box: Locally assigned symbols display in black. Symbols that inherit their values display in blue.

symbolname Equals symbolvalue

You can define symbols and assign values to them by typing them into the test plan, using this syntax:

```
$symbolname =symbolvalue
```

where `symbolname` is any valid 4Test identifier name, prefixed with the `$` character and `sybmolvalue` is any string, list, array, or the ? character (which indicates an undefined value).

For example, the following statement defines a symbol named Color and assigns it the `STRING` value "Red":

```
$Color = "Red"
```

If a symbol is defined at a level in the plan where it can be shared by a group of tests, each test can assign its own local value to the symbol, overriding whatever value it had at the higher level.

Specifying Symbols as Arguments when Entering a testcase Statement

1. Place the insertion cursor in the test plan at the location where the `testcase` statement is to be inserted. Placing a symbol name in the argument list of a `testcase` statement only specifies the name of the symbol; you also need to define the symbol and assign it a value at either the group or test case description level, as appropriate.

   If you do not know the value when you are initially writing the test plan, assign a question mark (?) to avoid getting a compiler error when you compile the test plan; doing so will also cause the tests to be counted as incomplete when a **Completion report** is generated.
2. Click **Testplan** > **Detail**.
3. Enter the name of a data driven test case on the **Testplan Detail** dialog box, followed by the argument list enclosed in parenthesis. If the test case expects a record, and not individual values, you must use the list constructor operator (curly braces).
4. Click **OK**.
   SilkTest Classic dismisses the **Testplan Detail** dialog box and inserts the `testcase` statement into the test plan.

Attributes and Values

This section describes site-specific characteristics that you can define for your test plan and assign to test descriptions and group descriptions.

Overview of Attributes and Values

Attributes are site-specific characteristics that you can define for your test plan and assign to test descriptions and group descriptions. Attributes are used to categorize tests, so that you can reference them as a group. Attributes can also be incorporated into queries, which allow you to mark tests that match the query's criteria. Marked tests can be run as a group.

By assigning attributes to parts of the test plan, you can:

- Group tests in the plan to distinguish them from the whole test plan.
- Report on the test plan based on a given attribute value.
- Run parts of the test plan that have a given attribute value.

For example, you might define an attribute called Engineer that represents the set of QA engineers that are testing an application through a given test plan. You might then define values for Engineer like David,

Jesse, Craig, and Zoe, the individual engineers who are testing this plan. You can then assign the values of Engineer to the tests in the test plan. Certain tests are assigned the value of David, others the value of Craig, and so on. You can then run a query to mark the tests that have a given value for the Engineer attribute. Finally, you can run just these marked tests.

Attributes are also used to generate reports. For example, to generate a report on the number of passed and failed tests for Engineer Craig, simply select this value from the **Pass/Fail Report** dialog box. You do not need to mark the tests or build a query in this case.

Attributes and values, as well as queries, are stored by default in `testplan.ini` which is located in the SilkTest Classic installation directory. The initialization file is specified in the **Data File for Attributes and Queries** field in the **General Options** dialog box.

SilkTest Classic ships with predefined attributes. You can also create up to 254 user-defined attributes.

Make sure that all the QA engineers in your group use the same initialization body file. You can modify the definition of an attribute.

Modifying attributes and values through the **Define Attributes** dialog box has no effect on existing attributes and values already assigned to the test plan. You must make the changes in the test plan yourself.

Predefined Attributes

The test plan editor has three predefined attributes:

Developer	Specifies the group of QA engineers who developed the test cases called by the test plan.
Component	Specifies the application modules to be tested in this test plan.
Category	Specifies the kinds of tests used in your QA Department, for example, Smoke Test.

User Defined Attributes

You can define up to 254 attributes. You can also rename the predefined attributes.

The rules for naming attributes include:
- Attribute names can be up to 11 characters long.
- Attribute and value names are not case sensitive.

Adding or Removing Members of a Set Attribute

Tests can be assigned more than one value at a time for attributes whose type is `Set`.

For example, you might have a `Set` variable called *RunWhen* with three values: `UI`, `regression`, and`smoke`. You can assign any combination of these three values to a test or group of tests. Separate each value with a semicolon.

You can use the + or – operator to add or subtract elements to what were previously assigned.

Consider the following examples:

Using + to add numbers

```
RunWhen: UI; regression Test 1    testcase: t1 RunWhen: + smoke
Test 2    testcase: t2
```

In this example, Test 1 has the values `UI` and `regression` . The statement

```
RunWhen: + smoke
```

adds the value `smoke` to the previously assigned values, so Test 2 has the values `UI`, `regression`, and `smoke`.

Using - to remove numbers

```
RunWhen: UI; regression Test 1 testcase: t1 RunWhen: -
regression Test 2
```

testcase: t2

In this example, Test 1 has the values `UI` and `regression`. The statement

```
RunWhen: - regression
```

removes the value `regression` from the previously assigned values, so Test2 has the value `UI`.

Rules for Using + and -

- You must follow the + or – with a space.
- You can add or remove any number of elements with one statement. Separate each element with a semicolon.
- You can specify + elements even if no assignments had previously been made. The result is that the elements are now assigned.
- You can specify – elements even if no assignments had previously been made. The result is that the set's complement is assigned. Using the previous example, specifying:

  ```
  RunWhen: - regression
  ```

  when no `RunWhen` assignment had previously been made results in the values `UI` and `smoke` being assigned.

Defining an Attribute and its Values

1. Click **Testplan** > **Define Attributes**, and then click **New**.
2. Name the attribute.
3. Select one of the following types, and then click **OK**.

   Normal You specify values when you define the attribute. Users of the attribute in a test plan pick one value from the list.

   Edit You don't specify values when you define the attribute. Users type their own values when they use the attribute in a test plan.

   Set Like normal, except that users can pick more than one value.
4. On the **Define Attributes** dialog box, if you:
   - have defined an `Edit` type attribute, you are done. Click **OK** to close the dialog box.
   - are defining a `Normal` or `Set` type attribute, type a value in the text box and click **Add**.

Once attributes have been defined, you can modify them.

Assigning Attributes and Values to a Test Plan

Attributes and values have no connection to a test plan until you assign them to one or more tests using an assignment statement. To add an assignment statement, you can do one of the following:

- Type the assignment statement yourself directly in the test plan.
- Use the **Testplan Detail** dialog box.

Format

An assignment statement consists of the attribute name, a colon, and a valid attribute value, in this format:

```
attribute-name: attribute value
```

For example, the assignment statement that associates the `Searching` value of the *Module* attribute to a given test would look like:

```
Module: Searching
```

Attributes of type `Set` are represented in this format:

```
attribute-name: attribute value; attribute
value; attribute value; ...
```

Placement

Whether you type an assignment statement yourself or have the **Testplan Detail** dialog box enter it for you, the position of the statement in the plan is important.

To have an assignment statement apply to	Place it directly after the
An individual test	test description
A group of tests	group description

Assigning an Attribute from the Testplan Detail Dialog Box

1. Place the cursor in the test plan where you would like the assignment statement to display, either after the test description or the group description.
2. Click **Testplan** > **Detail**, and then click the **Test Attributes** tab on the **Testplan Detail** dialog box.
   The arrow in the list box at the top of the dialog box identifies the test description at the cursor position in the test plan. The attribute will be added to this test description. The **Test Attributes** tab lists all your current attributes at this level of the test plan.
3. Do one of the following:
   - If the attribute is of type *Normal*, select a value from the list.
   - If the attribute is of type *Set*, select on or more values from the list.
   - If the attribute is of type *Edit*, type a value.
4. Click **OK**.
   SilkTest Classic closes the dialog box and places the assignment statements in the test plan.

Modifying the Definition of an Attribute

Be aware that modifying attributes and values through the **Define Attributes** dialog box has no effect on existing attributes and values already assigned to the test plan. You must make the changes in the test plan yourself.

1. Click **Testplan** > **Define Attributes**.
2. On the **Define Attributes** dialog box, select the attribute you want to modify, then:

Rename an attribute	Edit the name in the **Name** text box.
Assign a new value to the attribute	Type the value in the text box at the bottom right of the dialog box, and click **Add**. The value is added to the list of values.

Modify a value	Select the value from the **Values** list box, and click **Edit**. The value displays in the text box at the bottom right of the dialog box and the **Add** button is renamed to **Replace**. Modify the value and click **Replace**.
Delete a value	Select the value from the **Values** list box and click **Remove**. The text box is cleared and the value is removed from the **Values** list box.
Delete an attribute	Click **Delete**.

3. Click **OK**.
   The attributes and values are saved in the initialization file specified in the **General Options** dialog box.

Queries

This section describes how you can use a test plan query to mark all tests that match a user-selected set of criteria, or test characteristics.

Overview of Test Plan Queries

You can use a test plan query to mark all tests that match a user-selected set of criteria, or test characteristics. A query comprises one or more of the following criteria:

- Test plan execution: script file, test case name, or test data
- Test attributes and values
- Symbols and values

Test attributes and symbols must have been previously defined to be used in a query.

Named queries are stored by default in `testplan.ini`. The initialization file is specified in the **Data File for Attributes and Queries** text box in the **General Options** dialog box. The `testplan.ini` file is in the SilkTest Classic installation directory. Make sure that all the QA engineers in your group use the same initialization file.

Overview of Combining Queries to Create a New Query

You can combine two or more existing queries into a new query using the **Mark by Named Query** dialog box. The new query can represent the union of the constituent queries (logical OR) or the intersection of the constituent queries (logical AND).

Combining by union

Combining two or more queries by union creates a new named query that marks all tests that would have been marked by running each query one after the other while retaining existing marks. Since **Mark by Named Query** clears existing marks before running a query, the only way to achieve this result is to create a new query that combines the constituent queries by union.

Example

Suppose you have two queries, Query1 and Query2, that you want to combine by union.

Query1	Query2
Developer: David	Developer: Jesse
Component: Searching	TestLevel: 2

The new query created from the union of Query1 and Query2 will first mark those tests that match all the criteria in Query1 (Developer is David and Component is Searching)

and then mark those tests that match all the criteria in Query2 (Developer is Jesse and TestLevel is 2).

Combining by intersection

Combining two or more queries by intersection creates a new named query that marks every test that has the criteria specified in all constituent queries.

Example

For example, combining Query1 and Query2 by intersection would create a new query that comprised these criteria: Developer is David and Jesse, Component is Searching, and TestLevel is 2. In this case, the new query would not mark any tests, since it is impossible for a test to have two different values for the attribute Developer (unless Developer were defined as type Set under Windows). Use care when combining queries by intersection.

Guidelines for Including Symbols in a Query

- Use ? (question mark) to indicate an unset value. For example, `Mysymbol = ?` in a query would mark those tests where *Mysymbol* is unset. Space around the equals sign (=) is insignificant.
- If you need to modify the symbol in the query, select it from the list box and click **Edit**. The test plan editor places it in the text box and changes the **Add** button to **Replace**. Edit the symbol or value and click **Replace**.
- To exclude the symbol from the query, select it from the list box and click **Remove**. The test plan editor deletes it from the list box.

The Differences between Query and Named Query Commands

Testplan > **Mark by Query** or **Testplan** > **Mark by Named Query** both create queries, however, **Mark by Named Query** provides extra features, like the ability to combine queries or to create a query without running it immediately. If the query-creation function and the query-running function are distinct in your company, then use **Mark by Named Query**. If you intend to run a query only once, or run a query while keeping existing marks, then use **Mark by Query**.

The following table highlights the differences between the two commands.

Mary by Query	Mark by Named Query
Builds a query based on criteria you select and runs query immediately.	Builds a new query based on criteria you select. Can run query at any time.
Name is optional, but note that only named queries are saved and can be rerun at any time in the **Mark by Named Query** dialog box.	Name is required. Query is saved.
Cannot edit or delete a query.	Can edit or delete a query.
Cannot combine queries.	Can combine queries into a new query.
Lets you decide whether or not to clear existing marks before running new query. Unmarks by default.	Clears existing marks before running new query.

Unnamed queries can be run only once. If you name the query, you can have the test plan editor run it in the same or subsequent work sessions without having to rebuild the query or manually remark the tests that you're interested in rerunning or reporting on.

Create a New Query

You can create a new query through either **Testplan** > **Mark by Query** or **Testplan** > **Mark by Named Query**. You can also create a new query by combining existing queries.

1. Open the test plan and any associated sub-plans.
2. Click **Testplan** > **Mark by Query** or **Testplan** > **Mark by Named Query**.
3. Identify the criteria you want to include in the query. To include:
   - A script, test case, or test data, use the **Test Execution** tab. Use the **Script** and **Testcase** buttons to select a script and test case, or type the full specification yourself. To build a query that marks only manual tests, enter the keyword manual in the **Testcase** text box.
   - Existing attributes and values in the query, use the **Test Attributes** tab.
   - One or more existing symbols and values, use the **Symbols** panel. Type the information and click **Add**. The symbol and value are added to the list box.

   Do not type the dollar sign ($) prefix before the symbol name. The wildcard characters * (asterisk) and ? (question mark) are supported for partial matches: * is a placeholder for 0 or more characters, and ? is a placeholder for 1 character.

   Example 1

   If you type `find_5 (*` in the **Testcase** field, the query searches all the `testcase` statements in the plan and marks those test descriptions that match, as well as all subordinate descriptions to which the matching `testcase` statement applies (those where the find_5 testcase passed in data).

   Example 2

   If you type `find.t` in the Script field, the query searches all `script` statements in the plan and marks those test descriptions that match exactly, as well as all subordinate descriptions to which the matching `script` statement applies (those in which you had specified find.t exactly). It would not match any `script` statements in which you had specified a full path.
4. Take one of the following actions, depending on the command you chose to create the query:

Mark by Query	Click **Mark** to run the query against the test plan. The test plan editor closes the dialog box and marks the test plan, retaining the existing marks if requested.
Mark by Named Query	Click **OK** to create the query. The **New Testplan Query** dialog box closes, and the **Mark by Named Query** dialog box is once again visible. The new query displays in the **Testplan Queries** list box. If you want to: • Run the query, select it from the list box and click **Mark**. • Close the dialog box without running the query, click **Close**.

Edit a Query

1. Click **Testplan** > **Mark by Named Query** to display the **Mark by Named Query** dialog box.
2. Select a query from the **Testplan Queries** list box and click **Edit**.
3. On the **Edit Testplan Query** dialog box, edit the information as appropriate, and then click **OK** .
4. To run the query you just edited, select the query and click **Mark** . To close the dialog box without running the edited query, click **Close** .

Delete a Query

1. Click **Testplan** > **Mark by Named Query** to open the **Mark by Named Query** dialog box.
2. Select a query from the **Testplan Queries** box and click **Remove**.
3. Click **Yes** to delete the query, and then click **Close** to close the dialog box.

Combining Queries

1. Click **Testplan** > **Mark by Named Query** to display the **Mark by Named Query** dialog box.
2. Click **Combine**.
   The **Combine Testplan Queries** dialog box lists all existing named queries in the **Queries to Combine** list box.
3. Specify a name for the new query in the **Query Name** text box.
4. Select two or more queries to combine from the **Queries to Combine** list box.
5. Click the option button that represents the combination method to use: either **Union of Queries** or **Intersection of Queries**.
6. Click **OK** to save the new query.
   The **Mark by Named Query** dialog box displays with the new query in the **Testplan Queries** list box.
7. To run the query, select the query and click **Mark** or click **Close** to close the dialog box without running the query.

Designing and Recording Testcases

Hierarchical Object Recognition

When you record window declarations, SilkTest records descriptions based on hierarchical object recognition of the GUI objects in your application. SilkTest stores the declarations in an include file (*.inc). When you record or replay a testcase, SilkTest 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, SilkTest 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, SilkTest 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 the SWT sample application, SilkTest 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, Newmenu item, and xWindow menu item. For example:

```
Menu File
  tag "File"
  MenuItem New
```

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

Dynamic Object Recognition

Dynamic object recognition enables you to create test cases that use XPath queries to find and identify objects. Dynamic object recognition uses a `Find` or `FindAll` method to identify an object in a test case. For example, the following query finds the first top-level Shell with the caption SWT Test Application:

```
Desktop.find("/Shell[@caption='SWT Test Application']")
```

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

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

- In any application environment where the graphical user interface is undergoing changes. For example, to test the Check Me check box in a dialog box that belongs to a menu where the menu and the dialog box name are changing, using dynamic object recognition enables you to test the check box without concern for what the menu and dialog box are called. You can then verify the check box name, dialog box name, and menu name to ensure that you have tested the correct component.
- In a Web application that includes dynamic tables or text. For example, to test a table that displays only when the user points to a certain item on the web page, use dynamic object recognition to have the test case locate the table without regard for which part of the page needs to be clicked in order for the table to display.
- In an Eclipse environment that uses views. For example, to test an Eclipse environment that includes a view component, use dynamic object recognition to identify the view without regard to the hierarchy of objects that need to open prior to the view.

Using dynamic object recognition compared to using hierarchical object recognition

The benefits of using dynamic object recognition rather than hierarchical object recognition include:

- Dynamic object recognition uses a subset of the XPath query language, which is a common XML-based language defined by the World Wide Web Consortium, W3C. Hierarchical object recognition is based on the concept of a complete description of the application's object hierarchy and as a result is less flexible than dynamic object recognition.
- Dynamic object recognition requires a single object rather than an include file that contains window declarations for the objects in the application that you are testing. Using XPath queries, a test case can locate an object using a `Find` command followed by a supported XPath construct. Hierarchical object recognition uses the include file to identify the objects within the application.

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.

Using dynamic object recognition and window declarations

SilkTest Classic provides an alternative to using `Find` or `FindAll` functions in scripts that use dynamic object recognition. By default, when you record a test case with the Open Agent, SilkTest Classic uses locator keywords in an include (.inc) file to create scripts that use dynamic object recognition and window declarations. Using locator keywords with dynamic object recognition enables users to 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, SilkTest Classic tag-based scripts and leverage the power of XPath queries.

Existing test cases that use dynamic object recognition without locator keywords in an INC file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object

recognition without locator keywords in an INC file. You must manually record test cases that use dynamic object recognition without locator keywords. You can record the XPath query strings to include in test cases by using the **Locator Spy** dialog box.

Highlighting Objects During Recording

During recording, the active object in the AUT is highlighted by a green rectangle. As soon as a new object becomes active this new object is highlighted. If the same object remains active for more than 0.5 seconds a tool-tip will be displayed that displays the class name of the active object and also the current position of the mouse relative to the active object. This tool-tip will no longer be displayed when a new object becomes active, the user presses the mouse, or automatically after 2 seconds.

Overview of the Locator Keyword

Traditional SilkTest Classic scripts that use the Classic Agent use hierarchical object recognition. When you record a script that uses hierarchical object recognition, SilkTest 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, SilkTest 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. SilkTest 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 advantage 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, SilkTest 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, SilkTest 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. SilkTest Classic automatically expands the syntax to use full XPath strings when you run a script. You can omit:

- The hierarchy separator, “.//”. SilkTest Classic defaults to using “//”.
- The class name. SilkTest 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.

Setting Recording and Replay Options

This section describes how you can set options to optimize recording and replay.

Setting Recording Preferences

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

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

1. Click **Options** > **Recorder**.
   The **Recording Options** dialog box opens.
2. To set **Ctrl+Shift** as the shortcut key combination to use to pause recording, check the **OPT_ALTERNATE_RECORD_BREAK** check box.

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

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

   By default this value is set to 200.
6. Click **OK**.

Setting Recording Options for xBrowser

There are several options that can be used to optimize the recording of Web applications.

1. Click **Options** > **Recorder**.
2. Check the **Record mouse move actions** box if you are testing a web page that uses mouse move events.

   SilkTest Classic will only record mouse move events that cause changes to the hovered element or its parent in order to keep scripts short.

3. You can change the **mouse move delay** if required.

   Mouse move actions will only be recorded if the mouse stands still for this time. A shorter delay will result in more unexpected mouse move actions.
4. Click the **Browser** tab.
5. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording.

   For example, if you do not want to record attributes named **height**, add the **height** attribute name to the grid. Separate attribute names with a comma.
6. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording.

   For example, if you do not want to record attributes assigned the value of **x-auto**, add **x-auto** to the grid. Separate attribute values with a comma.
7. To record native user input instead of DOM functions, check the **OPT_XBROWSER_RECORD_LOWLEVEL** check box.

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

   If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.
8. Click the **Custom Attributes** tab.
9. Select **xBrowser** in the **Select a tech domain** list box and add the DOM attributes that you want to use for locators to the text box.

   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. 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, SilkTest Classic uses the default attributes for xBrowser.
10. Click **OK**.

    You can now record or manually create a test that uses ignores browser attributes and uses the type of page input that you specified.

Setting Custom Attributes to Use in Locators

The Open Agent includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results. 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” />`. This approach can eliminate the maintenance associated with locator changes. 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 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 boxes, both text boxes will return when you call the loginName attribute.

To configure custom attributes for locators:

1. Click **Options** > **Recorder** and then click the **Custom Attributes** tab.
2. 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.
3. 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, SilkTest Classic uses the default attributes for the application that you are testing.

   Separate attribute names with a comma.

   The following examples describe how you can include attributes in different application types:

   - To include custom attributes in a Web application, add them to the html tag. Type `<input type='button' bcauid='abc' value='click me' />` to add an attribute called bcauid.
   - To include custom attributes in a Java SWT application, use the `org.swt.widgets.Widget.setData`(String <varname>key</varname>, Object <varname>value</varname>) method.
   - To include custom attributes in a Swing application, use the `SetClientProperty`("propertyName", "propertyValue") method.
4. Click **OK**.
   You can now record or manually create a test case.

Setting Classes to Ignore

To specify the names of any classes that you want to ignore during recording and replay:

1. Click **Options** > **Recorder**.
   The **Recording Options** dialog box opens.
2. Click the **Transparent Classes** tab.
3. In the **Transparent classes** grid, type the name of the class that you want to ignore during recording and replay.

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

Setting WPF Classes to Expose During Recording and Playback

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

Specify the names of any WPF classes that you want to expose during recording and playback. For example, if a custom class called MyGrid derives from the WPF `Grid` class, the objects of the MyGrid custom class are not available for recording and playback. `Grid` objects are not available for recording and playback because the `Grid` class is not relevant for functional testing since it exists only for layout

purposes. As a result, `Grid` objects are not exposed by default. In order to use custom classes that are based on classes that are not relevant to functional testing, add the custom class, in this case MyGrid, to the OPT_WPF_CUSTOM_CLASSES option. Then you can record, playback, find, verify properties, and perform any other supported actions for the specified classes.

1. Click **Options** > **Recorder**.
   The **Recording 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 Pre-Fill During Recording and Replaying

You can define whether items in a WPFItemsControl, like WPFComboBox or WPFListBox, are pre-filled during recording and playback. WPF itself lazily loads items for certain controls, so these items are not available for SilkTest Classic if they are not scrolled into view. Turn pre-filling on, which is the default setting, to additionally access items that are not accessible without scrolling them into view. However, some applications have problems when the items are pre-filled by SilkTest Classic in the background, and these applications can therefore crash. In this case turn pre-filling off.

1. Click **Options** > **Recorder**.
   The **Recording Options** dialog box opens.
2. Click the **WPF** tab.
3. In the **Prefill items** area, check the **OPT_WPF_PREFILL_ITEMS** check box.
4. Click **OK**.

Setting Replay Options for the Open Agent

There are several options that can be used to optimize replaying applications.

1. Click **Options** > **Agent**.
2. Click the `Replay` tab.
3. From the **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 window is active before a call is executed, check the **Ensure window is active** check box.
5. Click **OK**.

Creating Test Cases with the Open Agent

This section describes how you can use the Open Agent to create test cases.

Application Configuration

An application configuration defines how SilkTest connects to the application that you want to test. SilkTest automatically creates an application configuration when you create the base state. However, at times, you

might need to modify, remove, or add an additional application configuration. For example, if you are testing an application that modifies a database and you use a database viewer tool to verify the database contents, you must add an additional application configuration for the database viewer tool.

An application configuration includes the:

- Executable pattern, which is the executable name for the application.

  For example, the executable pattern for Notepad is "*notepad.exe" (note the asterisk).
- Technology domains that you want to use for the application framework or technology.

  For example, the technology domains for the sample Adobe Flex dashboard application are WIN32 and Flex.
- Command line pattern, which is optional.

  A command line pattern creates an application configuration that matches the executable pattern and the command line pattern. An application configuration that contains a command line pattern enables only processes for testing that match both the executable pattern and the command line pattern. If no command-line pattern is defined, all processes with the specified executable pattern are enabled. Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, "*\javaw.exe" is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you is want is enabled for testing. For example, if the command line of the application ends with com.example.MyMainClass you might want to use *com.example.MyMainClass as the command line pattern.

Recording Test Cases With the Open Agent

When you record a test case with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations. With this approach, 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, SilkTest tag-based scripts and leverage the power of XPath queries.

1. Click **Record Testcase** on the basic workflow bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
   The **Record Testcase** dialog box opens.
2. Type the name of your test case in the **Testcase name** text box.

   Test case names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
3. From the **Application State** list box, select **DefaultBaseState** to have the built-in recovery system restore the default base state before the test case begins executing.

   - If you choose DefaultBaseState as the application state, the test case is recorded in the script file as `testcase testcase_name ()`.
   - If you chose another application state, the test case is recorded as `testcase testcase_name () appstate appstate_name`.
4. If you do not want SilkTest to display the status window during playback when driving the application to the specified base state, uncheck the **Show AppState status window** check box.

   Typically, you check this check box. However, in some circumstances it is necessary to hide the status window. For instance, the status bar might obscure a critical control in the application you are testing.
5. Click **Start Recording**.
   SilkTest performs the following actions:

   - Closes the **Record Testcase** dialog box.
   - Starts your application, if it was not already running.
   - Removes the editor window from the display.
   - Displays the **Recording status** window.

- Waits for you to take further action.

6. Interact with your application, driving it to the state that you want to test.

   As you interact with your application, SilkTest records your interactions in the **Testcase Code** box of the **Record Testcase** dialog box, which is not visible.

7. To add a validation to the script, press **Ctrl+Alt**.

   Note: For any application that uses **Ctrl+Shift** as the shortcut key combination, press **Ctrl+Shift** to open the **Verify Properties** dialog box.

   The **Verify Properties** dialog box opens.

8. Check the check boxes for the properties that you want the script to verify and then click **Ok**.
   The recorder adds a verification action to the script for the locator to which you pointed when you pressed the shortcut key combination, **Ctrl+Alt** or **Ctrl+Shift**.

9. To review what you have recorded, click **Stop Recording** in the **Recording** window.
   SilkTest displays the **Record Testcase** dialog box, which contains the code that has been recorded for you.

10. To resume recording your interactions, click **Resume Recording** in the dialog box. To temporarily suspend recording, click **Stop Recording** on the **Recording** window.

11. Click **Paste to Editor**.
    If you have interacted with objects in your application that have not been identified in your include files, the **Update Files** dialog box opens.

12. Perform one of the following steps:

    - Click **Paste testcase and update window declaration(s)** and then click **OK**. In most cases, you want to choose this option.
    - Choose **Paste testcase only** and then click **OK**. This option does not update the window declarations in the INC file when it pastes the script to the Editor. If you previously recorded the window declarations related to this test case, choose this option.

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**.

Recording Locators Using the Locator Spy

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

To record locators with the Locator Spy, you must use the Open Agent.

To record a locator using the Locator Spy:

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 and 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.
   - If you use Picking mode, click the object that you want to record and press the Record Break keys.
6. 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.

   SilkTest 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 SilkTest 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. To replace the locator that you recorded, select the locator that you want to use in the **Locator Details** table.

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

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

Specifying Whether To Use Locators or Tags To Resolve Window Declarations

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

1. Click **Options** > **General**.
   The **General Options** dialog box opens.
2. Specify if you want to use locators or tags to resolve window declarations.
   - To use locators to resolve window declarations, check the **Prefer Locator** check box.
   - To use tags to resolve window declarations, uncheck the **Prefer Locator** check box.
3. Click **OK**.

Setting Custom Attributes to Use in Locators

The Open Agent includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results. 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” />`. This approach can eliminate the maintenance associated with locator changes. 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 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 boxes, both text boxes will return when you call the loginName attribute.

To configure custom attributes for locators:

1. Click **Options** > **Recorder** and then click the **Custom Attributes** tab.
2. 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.
3. 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, SilkTest Classic uses the default attributes for the application that you are testing.

Separate attribute names with a comma.

The following examples describe how you can include attributes in different application types:

- To include custom attributes in a Web application, add them to the html tag. Type `<input type='button' bcauid='abc' value='click me' />` to add an attribute called bcauid.
- To include custom attributes in a Java SWT application, use the `org.swt.widgets.Widget.setData`(String <varname>key</varname>, Object <varname>value</varname>) method.
- To include custom attributes in a Swing application, use the `SetClientProperty`("propertyName", "propertyValue") method.

4. Click **OK**.
   You can now record or manually create a test case.

Configuring Applications

When you configure an application, SilkTest automatically creates a base state for the application. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution.

SilkTest has slightly different procedures depending on whether you are configuring a Web application or an application that does not use a Web browser, such as a Windows application, Java SWT application, or an Adobe Flex application.

XPath Basic Concepts

SilkTest 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.

SilkTest 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

SilkTest 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 SilkTest supports.

Supported XPath Construct	Sample	Description
Attribute	`MenuItem[@caption='abc']`	Finds all menu items with the given caption attribute in their object definition

Supported XPath Construct	Sample	Description
		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 SilkTest does not support.

Unsupported XPath Construct	Example
Comparing two attributes with each other.	`PushButton[@caption = @windowid]`

Unsupported XPath Construct	Example
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']"

Supported Attribute Types for Dynamic Object Recognition

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 custom attributes list values.

To create tests that use locators, you must use the Open Agent.

XPath Samples

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

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

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

Modifying an Application Configuration

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

1. Click **Options** > **Application Configurations**.
   The **Edit Application Configuration** dialog box opens and lists the existing application configurations.
2. To add an additional application configuration, click **Add** and then perform the following steps.
   Additional text boxes display so that you can add the new configuration.
3. In the **Executable Pattern** text box, type the executable name and file path of the application that you want to test. For example, you might type *\IEXPLORE.EXE to specify Internet Explorer.
4. To use a command line pattern in combination with the executable file, in the **Command Line Pattern** text box, type the command line pattern.
   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe`. This means that when you create an application configuration for a typical Java application, the executable pattern, *\javaw.exe is used, which matches any Java process. Use the command line pattern in such cases to ensure that only the application that you is want is enabled for testing. For example, if the command line of the application ends with com.example.MyMainClass you might want to use *com.example.MyMainClass as the command line pattern.
5. To remove an application configuration, click **Remove** next to the appropriate application configuration.
6. Click **OK**.

Reasons for Failure of Creating an Application Configuration

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

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

Issue	Reason	Solution
Time out	• The system is not fast enough. • The size of the memory of the system is to small.	Use a faster system or try to reduce the memory usage on your current system.

Issue	Reason	Solution
User Account Control (UAC) fails	The application under test is executed with administrator rights.	Manually start the recorder and the clients with administrator rights.
64-bit application	The application uses a 64-bit technology that is not yet supported.	Use the corresponding 32-bit application.
Command-line pattern	The command-line pattern is to specific. This issue occurs especially for Java. The replay may not work as intended.	Remove ambiguous commands from the pattern.

Test Cases

This section describes how you can use automated tests to address single objectives of a test plan.

Overview of testcases

A testcase is an automated test that addresses one objective of a testplan. A testcase:

- Drives the application from the initial state to the state you want to test.
- Verifies that the actual state matches the expected (correct) state. (Your QA department might use the term baseline to refer to this expected state.) This stage is the heart of the testcase.
- Cleans up the application, in preparation for the next testcase, by undoing the steps performed in the first stage.

In order for a testcase to function properly, the application must be in a stable state when the testcase begins to execute. This stable state is called the base state. The recovery system is responsible for maintaining the base state in the event the application fails or crashes, either during a tescases's execution or between testcases.

Each testcase is independent and it should perform its own setup, driving the application to the state you want to test, executing the testcase, and then returning the application to the base state. The testcase should not rely on the successful or unsuccessful completion of another testcase, and the order in which the testcase is executed should have no bearing on its outcome. If a testcase relies on a prior testcase to perform some setup actions, and an error causes the setup to fail or, worse yet, the application to crash, all subsequent tescases will fail because they cannot achieve the state where the test is designed to begin.

A testcase has a single purpose: a single test case should verify a single aspect of the application. When a testcase designed in this manner passes or fails, it is easy to determine specifically what aspect of the target application is either working or not working.

If a testcase contains more than one objective, many outcomes are possible. Therefore, an exception may not point specifically to a single failure in the software under test but rather to several related function points. This makes debugging more difficult and time-consuming and leads to confusion in interpreting and quantifying results. The result is an overall lack of confidence in any statistics that might be generated. But there are techniques you can use to perform more than one verification in a testcase.

Types of Testcases

SilkTest supports two types of testcases, depending on the type of application that you are testing. You can create testcases that use:

Hierarchical object recognition	This is a fast, easy method for creating scripts. This type of testing is supported for all application types.
Dynamic object recognition	This is a more robust and easy to maintain method for creating scripts. However, dynamic object recognition is only supported for applications that use the SilkTest Open Agent.

If you are using the SilkTest Open Agent, you can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements. You can use both recognition methods within a single testcase if necessary.

Anatomy of a basic testcase

A testcase is comprised of testcase keywords and object-oriented commands. You place a group of testcases for an application into a file called a script.

Each automated test for an application begins with the testcase keyword, followed by the name of the testcase. The testcase name should indicate the type of testing being performed.

The core of the testcase is object-oriented 4Test commands that drive, verify, and clean up your application. For example, consider this command:

```
TextEditor.File.New.Pick
```

The first part of the command, `TextEditor.File.New`, is the name of a GUI object. The last part of the command, `Pick`, is the operation to perform on the GUI object. The dot operator (`.`) delimits each piece of the command. When this command is executed at runtime, it picks the **New** menu item from the **File** menu of the Text Editor application.

Types of Testcases

There are two basic types of testcases:

- Level 1 tests, often called smoke tests or object tests, verify that an application's GUI objects function properly. For example, they verify that text fields can accept keystrokes and check boxes can display a check mark.
- Level 2 tests verify an application feature. For example, they verify that an application's searching capability can correctly find different types of search patterns.

You typically run Level 1 tests when you receive a new build of your application, and do not run Level 2 tests until your Level 1 tests achieve a specific pass/fail ratio. The reason for this is that unless your application's graphical user interface works, you cannot actually test the application itself.

Testcase Design

When defining test requirements, the goal is to vigorously test each application feature. To do so, you need to decide which set of inputs to a feature will provide the most meaningful test results.

As you design your testcases, you may want to associate data with individual objects, which can then be referenced inside testcases. You may find this preferable to declaring global variables or passing parameters to your testcases.

The type of data you decide to define within a window declaration will vary, depending on the type of testing you are doing. Some examples include:

- The default value that you expect the object to have when it displays.
- The tab sequence for each of a dialog box's child objects.

The following declaration for the **Find** dialog contains a list that specifies the tab sequence of the dialog's children.

```
window DialogBox Find
   tag "Find"
   parent TextEditor
   LIST OF WINDOW lwTabOrder = {...}
      FindWhat
      CaseSensitive
      Direction
      Cancel
```

For more information about the syntax to use for lists, see *LIST data type*.

Before you begin to design and record tescases, make sure that the built-in recovery system can close representative dialogs from your aplication window. See *Testing the recovery system's ability to close your application's dialogs*.

Constructing a testcase

This section explains the methodology you use when you design and record a testcase.

A testcase has three stages

Each testcase that you record should have the following stages:

1. Stage 1: The testcase drives the application from the initial state to the state you want to test.
2. Stage 2: The testcase verifies that the actual state matches the expected (correct) state. (Your QA department might use the term baseline to refer to this expected state.) This stage is the heart of the testcase.
3. Stage 3: The testcase cleans up the application, in preparation for the next testcase, by undoing the steps performed in stage 1.

Each testcase is independent

Each testcase you record should perform its own setup in stage 1, and should undo this setup in stage 3, so that the testcase can be executed independently of every other testcase. In other words, the testcase should not rely on the successful or unsuccessful completion of another testcase, and the order in which it is executed should have no bearing on its outcome.

If a testcase relies on a prior testcase to perform some setup actions, and an error causes the setup to fail or, worse yet, the application to crash, all subsequent testcases will fail because they cannot achieve the state where the test is designed to begin.

A testcase has a single purpose

Each testcase you record should verify a single aspect of the application in stage 2. When a testcase designed in this manner passes or fails, it’s easy to determine specifically what aspect of the target application is either working or not working.

If a testcase contains more than one objective, many outcomes are possible. Therefore, an exception may not point specifically to a single failure in the software under test but rather to several related function points. This makes debugging more difficult and time-consuming and leads to confusion in interpreting and quantifying results. The net result is an overall lack of confidence in any statistics that might be generated.

There are techniques you can use to do more than one verification in a testcase. See Performing more than one verification in a testcase.

A testcase starts from a base state

In order for a testcase to be able to function properly, the application must be in a stable state when the testcase begins to execute. This stable state is called the base state. The recovery system is responsible for maintaining the base state in the event the application fails or crashes, either during a testcase’s execution or between testcases.

DefaultBaseState

To restore the application to the base state, the recovery system contains a routine called DefaultBaseState that makes sure that:

- The application is running and is not minimized
- All other windows (for example, dialogs) are closed

- The main window of the application is active

If these conditions are not sufficient for your application, you can customize the recovery system.

Defining test requirements

When defining test requirements, the goal is to rigorously test each application feature. To do so, you need to decide which set of inputs to a feature will provide the most meaningful test results.

Data in testcases

What data does the feature expect

A user can enter three pieces of information in the Find dialog:

- The search can be case sensitive or insensitive, depending on whether the **Case Sensitive** check box is checked or unchecked.
- The search can be forward or backward, depending on whether the **Down** or **Up** radio button is selected.
- The search can be for any combination of characters, depending on the value entered in the **Find What** text field.

Create meaningful data combinations

To organize this information, it is helpful to construct a table that lists the possible combinations of inputs. From this list, you can then decide which combinations are meaningful and should be tested. A partial table for the **Find** dialog is shown below:

Case Sensitive	Direction	Search String
Yes	Down	Character
Yes	Down	Partial word (start)
Yes	Down	Partial word (end)
Yes	Down	Word
Yes	Down	Group of words
Yes	Up	Character
Yes	Up	Partial word (start)
Yes	Up	Partial word (end)
Yes	Up	Word
Yes	Up	Group of words

Constructing a testcase

This section explains the methodology you use when you design and record a testcase.

A testcase has three stages

Each testcase that you record should have the following stages:

1. Stage 1: The testcase drives the application from the initial state to the state you want to test.
2. Stage 2: The testcase verifies that the actual state matches the expected (correct) state. (Your QA department might use the term baseline to refer to this expected state.) This stage is the heart of the testcase.

3. Stage 3: The testcase cleans up the application, in preparation for the next testcase, by undoing the steps performed in stage 1.

Each testcase is independent

Each testcase you record should perform its own setup in stage 1, and should undo this setup in stage 3, so that the testcase can be executed independently of every other testcase. In other words, the testcase should not rely on the successful or unsuccessful completion of another testcase, and the order in which it is executed should have no bearing on its outcome.

If a testcase relies on a prior testcase to perform some setup actions, and an error causes the setup to fail or, worse yet, the application to crash, all subsequent testcases will fail because they cannot achieve the state where the test is designed to begin.

A testcase has a single purpose

Each testcase you record should verify a single aspect of the application in stage 2. When a testcase designed in this manner passes or fails, it's easy to determine specifically what aspect of the target application is either working or not working.

If a testcase contains more than one objective, many outcomes are possible. Therefore, an exception may not point specifically to a single failure in the software under test but rather to several related function points. This makes debugging more difficult and time-consuming and leads to confusion in interpreting and quantifying results. The net result is an overall lack of confidence in any statistics that might be generated.

There are techniques you can use to do more than one verification in a testcase. See Performing more than one verification in a testcase.

A testcase starts from a base state

In order for a testcase to be able to function properly, the application must be in a stable state when the testcase begins to execute. This stable state is called the base state. The recovery system is responsible for maintaining the base state in the event the application fails or crashes, either during a testcase's execution or between testcases.

DefaultBaseState

To restore the application to the base state, the recovery system contains a routine called DefaultBaseState that makes sure that:

- The application is running and is not minimized
- All other windows (for example, dialogs) are closed
- The main window of the application is active

If these conditions are not sufficient for your application, you can customize the recovery system.

Defining test requirements

When defining test requirements, the goal is to rigorously test each application feature. To do so, you need to decide which set of inputs to a feature will provide the most meaningful test results.

Data in testcases

What data does the feature expect

A user can enter three pieces of information in the Find dialog:

- The search can be case sensitive or insensitive, depending on whether the **Case Sensitive** check box is checked or unchecked.

- The search can be forward or backward, depending on whether the **Down** or **Up** radio button is selected.
- The search can be for any combination of characters, depending on the value entered in the **Find What** text field.

Create meaningful data combinations

To organize this information, it is helpful to construct a table that lists the possible combinations of inputs. From this list, you can then decide which combinations are meaningful and should be tested. A partial table for the **Find** dialog is shown below:

Case Sensitive	Direction	Search String
Yes	Down	Character
Yes	Down	Partial word (start)
Yes	Down	Partial word (end)
Yes	Down	Word
Yes	Down	Group of words
Yes	Up	Character
Yes	Up	Partial word (start)
Yes	Up	Partial word (end)
Yes	Up	Word
Yes	Up	Group of words

Saving testcases

When saving a testcase, SilkTest does the following:

- Saves a source file, giving it the `.t` extension; the source file is an ASCII text file, which you can edit.
- Saves an object file, giving it the `.to` extension; the object file is a binary file that is executable, but not readable by you.

For example: If you name a testcase (script file) `mytests` and save it, you will end up with two files: the source file `mytests.t`, in the location you specify, and the object file `mytests.to`.

To save a new version of a script's object file when the script file is in view-only mode, choose **File** > **Save Object File**.

Recording without window declarations

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you record a testcase against a GUI object for which there is no declaration (or if you want to write a testcase from scratch against such an object), SilkTest requires a special syntax to uniquely identify the GUI object because there is no identifier.

This special syntax is called a dynamic instantiation and is composed of the class and tag of the object. The general syntax of this kind of identifier is:

```
class("tag").class("tag"). ...
```

For example, if there is not a declaration for the **Find** dialog of the Text Editor application, the syntax required to identify the object looks like this:

```
MainWin("Text Editor|$C:\PROGRAM FILES\<SilkTest install directory>\SILKTEST
\TEXTEDIT.EXE").DialogBox("Find")
```

To create the dynamic tag, the recorder uses the multiple-tag settings that are stored in the **Record Window Declarations** dialog. In the example shown above, the tag for the Text Editor contains its caption as well as its window ID. For more information, see *Overview of tags*.

Overview of Application States

When testing an application, typically, you have a number of testcases that have identical setup steps. Rather than record the same steps over and over again, you can record the steps as an application state and then associate the application state with the relevant testcases.

An application state is the state you want your application to be in after the base state is restored but before you run one or more testcases. By creating an application state, you are creating reusable code that saves space and time. Furthermore, if you need to modify the Setup stage, you can change it once, in the application state routine.

At most, a testcase can have one application state associated with it. However, that application state may itself be based on another previously defined application state. For example, assume that::

- The testcase Find is associated with the application state Setup
- The application state Setup is based on the application state OpenFile
- The application state OpenFile is based on the built-in application state, DefaultBaseState
- SilkTest would execute the programs in this order:

  1. DefaultBaseState application state
  2. OpenFile application state
  3. Setup application state
  4. Find testcase

If a testcase is based on a single application state, that application state must itself be based on DefaultBaseState in order for the testcase to use the recovery system. Similarly, if a testcase is based on a chain of application states, the final link in the chain must be DefaultBaseState. In this way, SilkTest's built-in recovery system is still able to restore the application to its base state when necessary.

Behavior of an application state based on NONE

If an application state is based on the keyword `NONE`, SilkTest executes the application state twice: when the testcase with which it is associated is entered and when the testcase is exited.

On the other hand, if an application state is based on DefaultBaseState, SilkTest executes the application state only when the associated testcase is entered.

The following example code defines the application state InvokeFind as based on the NONE keyword and associates that application state with the testcase TestFind.

```
Appstate InvokeFind () basedon none
 xFind.Invoke ()
 print ("hello")

testcase TestFind () appstate InvokeFind
 print ("In TestFind")
 xFind.Exit.Click ()
```

When you run the testcase in SilkTest, in addition to opening the Find dialog, closing it, and reopening it, the testcase also prints:

```
hello
In TestFind
hello
```

The testcase prints hello twice because SilkTest executes the application state both as the testcase is entered and as it is exited.

Example A word processors feature

For purposes of illustration, this topic develops test requirements for the searching feature of the sample Text Editor application using the **Find** dialog. This topic contains the following:

- Determining what data the feature expects
- Creating meaningful data combinations
- Overview of recording the stages of a testcase

When a user enters the criteria for the search and clicks the Find Next button, the search feature attempts to locate the string. If the string is found, it is selected (highlighted). Otherwise, an informational message is displayed.

Determining what data the feature expects

A user can enter three pieces of information in the Find dialog:

- The search can be case sensitive or insensitive, depending on whether the **Case Sensitive** check box is set or unset.
- The search can be forward or backward, depending on whether the **Down** or **Up** radio button is selected.
- The search can be for any combination of characters, depending on the value entered in the **Find What** text field.

Creating meaningful data combinations

To organize this information, it is helpful to construct a table that lists the possible combinations of inputs. From this list, you can then decide which combinations are meaningful and should be tested. A partial table for the **Find** dialog is shown below:

Case Sensitive	Direction	Search String
Yes	Down	Character
Yes	Down	Partial word (start)
Yes	Down	Partial word (end)
Yes	Down	Word
Yes	Down	Group of words
Yes	Up	Character
Yes	Up	Partial word (start)
Yes	Up	Partial word (end)
Yes	Up	Word
Yes	Up	Group of words

Overview of recording the stages of a testcase

A testcase performs its actions in three stages. The following table illustrates these stages, describing in high-level terms the steps for each stage of a sample testcase that tests whether the Find facility is working.

Setup

1. Open a new document.
2. Type text into the document.
3. Position the text cursor either before or after the text, depending on the direction of the search.
4. Select **Find** from the Search menu.
5. In the **Find** dialog:
   - Enter the text to search for in the **Find What** field.
   - Select a direction for the search.
   - Make the search case sensitive or not.
   - Click **Find Next** to do the search.
6. Click **Cancel** to close the **Find** dialog.

Verify

Record a 4Test verification statement that checks that the actual search string found, if any, is the expected search string.

Cleanup

1. Close the document.
2. Click **No** when prompted to save the file.

After learning the basics of recording, you can record from within a testplan, which makes recording easier by automatically generating for you the links that connect the testplan to the testcase.

Recording Testcases with the Classic Agent

Overview of recording the stages of a testcase

A testcase includes several stages. The following table illustrates these stages, describing in high-level terms the steps for each stage of a sample testcase that tests whether the Find facility is working.

After learning the basics of recording, you can record from within the testplan file, which makes recording easier by automatically generating the links that connect the testplan to the testcase.

Setup and Record

1. Open a new document.
2. Type text into the document.
3. Position the text cursor either before or after the text, depending on the direction of the search.
4. Select **Find** from the **Search** menu.
5. In the **Find** dialog:
   a. Enter the text to search for in the **Find What** field.
   b. Select a direction for the search.
   c. Make the search case sensitive or not.
   d. Click **Find Next** to do the search.
6. Click **Cancel** to close the Find dialog.

Verify

Record a 4Test verification statement that checks that the actual search string found, if any, is the expected search string.

Cleanup

1. Close the document.
2. Click **No** when prompted to save the file.

Overview of Recording 4Test components

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you want to manually write some or most of your 4Test code, or if you want to add individual lines to an existing testcase, you can use the following recording tools:

Record/Actions

For example, when you are working with a script, you might want to leave the **Record Actions** dialog open. Any time you want to verify a GUI object, you can point to the object in your application and verify it.

You can also use the dialog to write a syntactically correct 4Test statement based on your manual interaction with your application. This eliminates the need to search through the documentation for the correct method and its arguments. Once the statement is recorded, the **Paste to Editor** button inserts the statement to your script.

Record/Window Identifiers

Similar to the `Actions` command, **Record/Window Identifiers** records the fully qualified name of the GUI object you are pointing at, which you can then insert into your script. This eliminates the need to bring up your test frame file to find the correct identifier for the object.

Record/Window Locations

It can be useful to know the position of certain objects, for example objects that are drawn (like tools on a toolbar) or drawing regions (in a CAD/CAM package, for example). To record the location of an object, use the Record Window Locations dialog. You can also add a window location to an existing window declaration.

Record/Class

If you are using ActiveX, Visual Basic, or Java classes (controls) that are not predefined, you can record the classes for use in your tests.

Recording a Testcase With the Classic Agent

When you record a testcase with the Classic Agent, SilkTest uses hierarchical object recognition, a fast, easy method to create scripts. However, testcases that use dynamic object recognition are more robust and easy to maintain. You can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements. You can use both recognition methods within a single testcase if necessary.

1. Enable extensions and set up the recovery system.
2. Click **Record Testcase** on the Basic Workflow bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
3. Type the name of your testcase in the Testcase name field of the **Record Testcase** dialog. Testcase names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.

4. Select **DefaultBaseState** in the **Application State** field to have the built-in recovery system restore the default BaseState before the testcase begins executing. If you chose **DefaultBaseState** as the application state, the testcase is recorded in the script file as: `testcase testcase_name ()`. If you chose another application state, the testcase is recorded as: `testcase testcase_name () appstate appstate_name`.
5. If you do not want SilkTest to display the status window it normally shows during playback when driving the application to the specified base state—perhaps because the status bar obscures a critical control in the application you are testing—uncheck the **Show AppState status window** check box.
6. Click **Start Recording**. SilkTest:

   - Closes the **Record Testcase** dialog.
   - Starts your application, if it was not already running.
   - Removes the editor window from the display.
   - Displays the **Record Status** window.
   - Waits for you to take further action
7. Interact with your application, driving it to the state that you want to test. As you interact with your application, SilkTest records your interactions in the **Testcase Code** field of the **Record Testcase** dialog, which is not visible.
8. To review what you have recorded, click the **Done** button in the **Record Status** window.

   SilkTest displays the **Record Testcase** dialog, which contains the 4Test code that has been recorded for you.
9. To resume recording your interactions, click the **Resume Recording** button in the dialog. To temporarily suspend recording, click the **Pause Recording** button on the **Record Status** window.
10. Verify the testcase.

Verifying a testcase

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The cornerstone of an automated test is the verification stage, in which the test verifies that the state of the application matches the expected (baseline) state. Using the recorder, you can record object-appropriate verification of your application's state, data, or appearance.

To record the verification stage

1. Continue with these steps after you record a testcase. Or, if you have previously recorded a verification statement in an existing testcase, choose **Record** > **Actions** to modify it.
2. Drive your application to the state that you want to verify and position the mouse cursor over the object.
3. Look at the **Record Status** window and make sure it is listing the object you want to verify. If so, press `Ctrl+Alt`. The **Verify Window** dialog appears over your application window. The **Window** field, in the top-left corner of the dialog, displays the name of the object you were pointing at when you pressed `Ctrl+Alt`.

   If the name in the Window field is incorrect, click **Cancel** to close the dialog and return to the application. Point to the object you want to verify and press `Ctrl+Alt` again.
4. If a script file is not the active window, SilkTest prompts you for a file name. If prompted, specify the name of either a new or an existing script file and click **OK**.
5. Choose to verify any of the following:

   - Properties of an object.
   - Appearance using a bitmap.
   - An object's state using built-in verification methods or other methods in combination with the built-in Verify function.

6. If you are writing a complete testcase, record the cleanup stage and paste the testcase into the script. If you have added a verification statement to an existing testcase, paste it into your script and close the Record Actions dialog.

Recording the cleanup stage and pasting the recording

After performing the verification, continue to interact with your application. This is the cleanup stage. For example, in the sample testcase, cleanup means closing the document window without saving it.

1. When you have finished recording your testcase or just want to see what you have recorded, click **Done** on the **Record Status** window. SilkTest displays the **Record Testcase** window again. The **Testcase Code** field contains your interactions written as 4Test code.
2. Review the code and take the following actions:
   - All the information in the window is complete and what you want, then click **Paste to Editor**. SilkTest closes the **Record Testcase** dialog and places the new testcase in your script file. If the 4Test code is not what you expected, then edit the name in the **Testcase Name** field.
   - If the testcase name is not what you want, then edit the name in the **Testcase Name** field.
   - If the application state is not the one you want, then delete the code in the **Testcase Code** field, select a new application state from the drop-down list and click **Resume Recording** to re-record the testcase.
   - If the testcase is not finished, then click **Resume Recording**. The **Record Testcase** window is reopened. You can continue to record your interactions.

   Note: When you paste a recorded testcase (or other recorded actions, such as when you use **Record Actions**) into a script, SilkTest indents the code under a recording statement to facilitate playback. For more information, see *recording statement*.
3. Click **Paste to Editor**. If you have interacted with objects in your application that have not been identified in your include files, the **Update Files** dialog opens. Choosing **Paste testcase only**, does not update any `.inc` files while it pastes to the script with dynamically instantiated new objects. **Update window declarations and testcase** will create window declarations for new objects and use the new identifiers in the resulting testcase. Note that if you edit the contents of the recorder window, then you must allow SilkTest to update the window declarations. The **Paste testcase only** option will be disabled.
4. Click **File** > **Save** to save the script file.

Testing the recovery systems ability to close your applications dialogs

Before you begin to design and record testcases, make sure that the built-in recovery system can close representative dialogs from your application. Although the recovery system is robust enough to be able to close almost any application window, some applications may have windows that close in an unconventional fashion.

Here are the three types of dialogs you should test:

- A modal dialog (a dialog that locks you out of the rest of your application until you dismiss it).
- A non-modal dialog.
- A non-modal dialog that causes the display of a confirmation dialog.

To test the recovery system's ability to close your application's dialogs

1. Start SilkTest.
2. If you have not already done so, record a test frame for your application.
3. Choose **Options** > **Runtime** to ensure that your application's test frame file is listed in the Use Files field in the Runtime Options dialog.

4. Start your application and invoke a representative dialog.
5. In SilkTest, choose **Run** > **Application State**.
6. On the **Run Application State** dialog, select the **DefaultBaseState application state** and click **Run**.
7. SilkTest executes the `DefaultBaseState` routine, which should close the dialog and any open windows, then display a results file.

   If the built-in recovery system cannot close one of the three representative dialogs, you need to modify the recovery system so that it understands how to close the dialog. For more information, see *Specifying new window closing procedures*.

Linking to a script and testcase by recording a testcase

1. Place the cursor at the end of a test description or a group description.
2. Choose **Record** > **Testcase**. SilkTest prompts you to name a script file to contain the testcase. SilkTest does not prompt you for a script file if there is a script defined at a higher level and inherited by the testcase you are recording. If that script exists, SilkTest puts the testcase in that script.
3. If prompted, select an existing script from the list or enter the name of a new script in the **File Name** field, then click **OK**.
4. On the **Record Testcase** dialog, type the name for the testcase and optionally select an application state to be run before the recording starts.
5. Click **Start Recording**. SilkTest displays the **Recording Status** dialog. The dialog flashes the word `Recording` for the duration of the session.
6. When you are finished recording the actions that comprise the testcase, click **Done** in the **Recording Status** dialog.
7. On the **Record Testcase** dialog, click **Paste to Editor**. SilkTest closes the **Record Testcase** dialog and inserts the testcase into the script file. It also adds the script and testcase statements to the testplan on a new line and indents them appropriately.

If the script file is inherited by the testcase you are recording, only the testcase statement is pasted.

Saving a script file

To save a script file, click **File** > **Save**. If it is a new file, SilkTest prompts you for the file name and location.

If you are working within a project, SilkTest prompts you to add the file to the project. Click **Yes** if you want to add the file to the open project, or **No** if you do not want to add this file to the project.

To save a new version of a script's object file when the script file is in view-only mode, choose **File** > **Save Object File**.

If you are working within a project, you can add the file to your project. If you add object files (`.to`, `.ino`) to your project, the files will appear under the **Data** node on the **Files** tab. You cannot modify object files within the SilkTest editor because object files are binary. To modify an object file, open the source file (`.t` or `.inc`), edit it, and then recompile.

Recording an application state

You define an application state before recording the testcase(s) associated with it. As with testcases, you can write an application state routine from scratch or you can use the Application State command on the Record menu.

1. Open the file in which you want to place the application state. This can either be the test frame file for the application or the script file where the associated testcases are defined. If you put the application

state in the test frame file, it will be available to all testcases. If you put it in the script file, it will be available only to testcases in that script file.

2. Open the application that you want to test.
3. Choose **Record** > **Application State** to display the **Record Application State** dialog.
4. Type the name of your new application state in the **Application State Name** field.
5. Select an application state from the **Based On list** box.
6. Click **Start Recording**. SilkTest closes the **Record Application State** dialog and displays the **Record Status** window. The **Status** field flashes `Recording`.
7. Drive your application to the state you want to record. At any point, you can record a verification by pressing `Ctrl+Alt`.
8. When you have finished recording an application state, click **Done** on the **Record Status** window. SilkTest redisplays the **Record Application State** dialog. The **Application State Code** field contains the 4Test code you recorded. You can take the following actions:

All the information in the window is complete and what you expect.	Click **Paste to Editor**. SilkTest closes the **Record Application State** dialog and places the new application state in your file.
You want to alter the code.	Edit the **Application State Code** field.
The application state name is not what you want.	Edit the name in the **Application State Name** field.
The application state on which this application state is based is not the one you want.	Delete the code in the **Application State Code** field, select a new application state from the drop-down list, and click **Resume Recording** to re-record the application state.
The application state routine is not finished.	Click Resume Recording. SilkTest opens the **Record Status** window.

Recording Actions

Use the **Record Actions** dialog to record the actions you perform to test an application. For example, you can also use the dialog to write a syntactically correct 4Test statement based on your manual interaction with your application. This eliminates the need to search through the documentation for the correct method and its arguments. Once the statement is recorded, the Paste to Editor button inserts the statement to your script.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Click **Record** > **Actions** to open the **Record Actions** dialog.
2. Perform the action that you want to record.

   The dialog displays the GUI object name when you point to an object. You can click **Pause Recording** to review the object properties that you have recorded. When you click **Resume Recording**, the status bar returns.
3. Press `Ctrl+Alt` to verify the action.
4. Click **Paste to Editor** and then click **Close**.

Recording the Location of an Object

You can record the x, y locations of a graphical control, such as a toolbar. It can be useful to know the position of certain objects, for example objects that are drawn (like tools on a toolbar) or drawing regions (in a CAD/CAM package, for example). A location is recorded relative to the screen, frame, and client window.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To record the location of an object:

1. Click **Record** > **Window Locations** to open the **Record Window Locations** dialog.
2. Position the cursor over the object that you want to record. The dialog displays the name of the object and its x,y coordinates relative to the screen, the frame (the main window and its window decoration), and the client (the main window minus its window decoration).
3. Press `Ctrl+Alt` to verify the location.
4. Click the option button that corresponds with the object that you want to record and what you intend to do with the recording.

   For example, if you plan to add the location of the toolbar to an existing window declaration, click the **Client** option.
5. Click **Paste to Editor** and then click **Close**.

Recording Window Identifiers

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can record window identifiers to record the fully qualified name of the GUI object you are pointing at in your test application, which you can then insert into your script.

If you are recording a test that uses hierarchical object recognition this eliminates the need to bring up your test frame file to find the correct identifier for the object.

1. Click **Record** > **Window Identifiers** to open the **Record Window Identifiers** dialog.
2. Position the cursor over the GUI object that you want to record.

   The text box displays the GUI object name when you point to an object.
3. If necessary, press `Ctrl+Alt` to pause recording the window identifier.

   Note: For any application that uses `Ctrl+Shift` as the shortcut key combination, press `Ctrl+Shift` to pause recording.
4. Select the identifier in the text box and choose one of the following:
5. Click **Close**.

Testing an application state

Before you run a testcase that is associated with an application state, make sure the application state compiles and runs without error.

1. Make active the window that contains the application state and choose **Run** > **Application State**.
2. On the **Run Application State** dialog, select the application state you want to run and click **Run**. If there are compilation errors, SilkTest displays an error window. Fix the errors and rerun the application state.

Verification

Overview of object properties

You will perform most of your verifications using properties. A `VerifyProperties` method statement is added to your script. The `VerifyProperties` method verifies the selected properties of an object and its children. See `VerifyProperties` method for more information.

Each object has many characteristics, or properties. For example, dialog boxes can have the following verification properties: `Caption`, `Children`, `DefaultButton`, `Enabled`, `Focus`, `Rect`, and `State`.

Caption is the text that appears in the dialog box's title bar. Children is a list of all the objects contained in the dialog box, **DefaultButton** is the button that is invoked when you press **Enter**, and so on. In your testcases, you can verify the state of any of these properties.

You can also, in the same testcase, verify properties of children of the selected object. For example, the child objects in the **Find** dialog box (such as the text field **FindWhat** and the check box **CaseSensitive**) will also be selected for verification.

By recording verification statements for the values of one or more of an object's properties, you can determine whether the state of the application is correct or in error when you run your testcases.

Verifying properties of an object

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Use this procedure to record verification statements to verify one or more characteristics, or properties, of an object.

1. Complete the steps in *Verifying a testcase*.
2. Click the **Properties** tab and then choose the objects to verify. To verify all or most objects, click **Check All** and then uncheck individual check boxes.
3. Choose the properties to verify in one of the following ways:
4. Click **OK** to close the **Verify Window** dialog.
5. If you are writing a complete testcase, record the cleanup stage and paste the testcase into the script. If you have added a verification statement to an existing testcase, paste it into your script and close the Record Actions dialog.

   Here are some points to note about the **Property** tab:

   - The **Windows to Verify** list box (left) displays the class and the identifier of all the objects whose properties have been captured. Indentation denotes the hierarchy. A checked check box (left margin) means that the object will be verified. By default, all objects are checked and the first object is selected.
   - The **Properties to Verify** list box (right) displays each property of the selected object and its current value. A checked check box (left margin) means that the property will be verified. By default, the properties of the selected property set (shown in the **Property Set** drop-down list box) are checked.
   - The **Property Value** field displays the value of the selected property. You can edit the value in this field if it is not what you want to verify against. The value specified in this field is the value you expect at runtime, that is, the baseline value.

Verifying an object using the Verify function

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Use this procedure to verify an object's state using built-in verification methods or other methods in combination with the built-in Verify function.

1. Complete the steps in *Verifying a testcase*.
2. On the **Verify Window** dialog, click the `Method` tab. SilkTest lists the methods for the selected class on the left.
3. Check the **Include Inherited** check box to see methods that the class inherits.
4. Select the method that will return the expected value and provide any needed arguments. You can specify a built-in method or a user-defined method (as long as it returns a value).
5. Click **OK**.
6. SilkTest returns you to the test application.

7. If you are writing a complete testcase, record the cleanup stage and paste the testcase into the script. If you have added a verification statement to an existing testcase, paste it into your script and close the **Record Actions** dialog.
8. In the editor, wrap the `Verify` function around the method that returns the expected value as follows: Make the method call the first argument, specify the expected value as the second argument, and provide an error message string optionally as the third argument.

For example, here is a testcase that verifies that the text in the `TextField Replace.FindWhat` is `myText`. It uses the built-in verification method `VerifyValue`.

```
testcase VerifyMethodTest ()
TextEditor.Search.Replace.Pick ()
Replace.FindWhat.VerifyValue
("myText")
Replace.Cancel.Click ()
```

Verifying object attributes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Each kind of GUI object in an application has a variety of characteristics, called attributes. For example, a text field has the following attributes:

- `Caret position`, which is the current position of the text insertion cursor, in (line, column) format. For example, a value of {1,1} means that the text insertion cursor is positioned on line 1, column 1.
- `Enabled`, which is the current enabled status of the text field, either true or false.
- `Selected range`, which is the beginning and ending position of the text string currently selected in the field, in (line, column) format. For example a value of {1,12,1,16} means that the selected text begins on line 1, column 12 and ends on line 1, column 16.
- `Selected Text`, which is the string that is currently selected, if any, in the text field.
- `Text`, which is the entire contents of the text field.

By recording verification statements for the values of one or more of an object's attributes, you can determine whether the state of the application is correct or in error when you run your testcases. That is: did the feature you are testing have the expected result?

By selecting the **Verify All Attributes** check box, you can record a test that verifies the state, contents, and value of a GUI object and any objects it contains. See *Verifying attributes of an object*. This is commonly called a smoke test or a Level 1 test. A smoke test uses the `VerifyEverything` method to verify every aspect of a particular GUI object.

If you need to, you can define and add your own attributes to the built-in hierarchy of GUI classes.

Attributes have been essentially rendered obsolete and have been replaced by properties. See *Verifying using properties*.

Verifying attributes of an object

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Click **Options** > **Recorder**. Uncheck the **Verify Using Properties** check box if necessary.

   You will not be able to uncheck this check box if you have enabled enhanced support for Visual Basic; it requires properties for verification.
2. Drive your application to the test state and press `Ctrl+Alt`. SilkTest displays the **Attribute** tab of the **Verify Window** dialog. The list box on the left shows the attributes for the current object.
3. Select an attribute from the list box or check the **Verify All Attributes** check box. In the **Attribute value** field SilkTest displays the current value of the attribute (that is, the value that exists when you are recording).

When verifying attributes during recording, the value size limit of the attribute is 256 characters. The name size limit is 32 characters. The total attribute value/name pair limit size is 4K. If the length exceeds 4K, the message `Unable to Get Windows Properties` is displayed.

4. If the current value of the attribute is not the value you want to test for at runtime, edit the **Attribute value** field. The value specified in this field is the value you expect at runtime, that is, the baseline value.
5. Click **OK** to accept the attribute and its value. SilkTest closes the **Verify Window** dialog and displays the **Record Status** window. The testcase will verify that the object has the attribute value selected. If not, SilkTest writes an error to the results file.

   With the **Verify Using Properties** check box unchecked, the next time you go to verify an object, the **Verify Window** dialog will have an **Attribute** tab, instead of a **Property** tab. For information on verification using properties, see *Verifying properties of an object*.

Overview of verifying bitmaps

A bitmap is a picture of some portion of your application. Verifying a bitmap is usually only useful when the actual appearance of an object needs to be verified to validate application correctness. For example, if you are testing a drawing or CAD/CAM package, a testcase might produce an illustration in a drawing region that you want to compare to a baseline. Other possibilities include the verification of fonts, color charts, and certain custom objects.

When comparing bitmaps, keep the following in mind:

- Bitmaps are not portable between GUIs. The format of a bitmap on a PC platform is `.bmp`.
- A bitmap comparison will fail if the image being verified does not have the same screen resolution, color, window frame width, and window position when the testcase is run on a different machine than the one on which the baseline image was captured.
- Make sure that your testcase sets the size of the application window to the same size it was when the baseline bitmap was captured.
- Capture the smallest possible region of the image so that your test is comparing only what is relevant.
- If practical, do not include the window's frame (border), since this may have different colors and/or fonts in different environments.

Verifying appearance using a bitmap

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Use this procedure to compare the actual appearance of an image against a baseline image. Or, use it to verify fonts, color charts, or custom objects.

1. Complete the steps in Verifying a testcase.
2. On the **Verify Window** dialog, click the **Bitmap** tab and then select the region to update: **Entire Window**, **Client Area of Window** (that is, without scroll bar or title bar), or **Portion of Window**.
3. In the **Bitmap File Name** field, type the full path of the bitmap file that will be created.

   The default path is based on the current directory. The default file name for the first bitmap is `bitmap.bmp`. Click **Browse** if you need help choosing a new path or name.
4. Click **OK**. If you selected **Entire Window** or **Client Area of Window**, SilkTest captures the bitmap and returns you to your test application. If you selected **Portion of Window**, position the cursor at the desired location to begin capturing a bitmap. While you press and hold the mouse button, drag the mouse to the screen location where you want to end the capture. Release the mouse button.

   A bitmap comparison will fail if the image being verified does not have the same screen resolution, color, window frame width, and window position as the baseline image.

   Capture the smallest possible region of the image so that your test is comparing only what is relevant.

5. If you are writing a complete testcase, record the cleanup stage and paste the testcase into the script. If you have added a verification statement to an existing testcase, paste it into your script and close the **Record Actions** dialog.

Overview of verifiying an objects state

Each class has a set of methods associated with it, including built-in verification methods. You can verify an object's state using one of these built-in verification methods or by using other methods in combination with the built-in Verify function.

A class's verification methods always begin with Verify. For example, a TextField has the following verification methods; VerifyPosition, VerifySelRange, VerifySelText, and VerifyValue.

You can use the built-in Verify function to verify that two values are equal and generate an exception if they are not. Typically, you use the Verify function to test something that does not map directly to a built-in property or method. Verify has the following syntax:

```
Verify (aActual, aExpected [, sDesc])
```

`aActual`

The value to verify. `ANYTYPE`.

`aExpected`

The expected value. `ANYTYPE`.

`sDesc`

`Optional`. A message describing the comparison. STRING.

Usually, the value to verify is obtained by calling a method for the object being verified; you can use any method that returns a value.

Example: verify an object

Say you want to verify the number of radio buttons in the **Direction RadioList** in the Replace dialog in the Text Editor. There is no property or method you can directly use to verify this. But there is a method for **RadioList**, `GetItemCount`, which returns the number of radio buttons. You can use the method to provide the actual value, then specify the expected value in the script, as follows:

When doing the verification, position the mouse pointer over the **RadioList** and press `Ctrl+Alt`. Click the **Method** tab in the **Verify Window** dialog, and select the `GetItemCount` method.

Click **OK** to close the **Verify Window** dialog, and complete your testcase. Paste it into a script. You now have the following script:

```
testcase VerifyFuncTest ()
TextEditor.Search.Replace.Pick ()
Replace.Direction.GetItemCount ()
Replace.Cancel.Click ()
```

Now use the `Verify` function to complete the verification statement. Change the line:

```
Replace.Direction.GetItemCount ()
```

to

```
Verify (Replace.Direction.GetItemCount (), 2)
```

That is, the call to `GetItemCount` (which returns the number of radio buttons) becomes the first argument to `Verify`. The expected value, in this case, 2, becomes the second argument.

Your completed script is:

```
testcase VerifyFuncTest ()
TextEditor.Search.Replace.Pick ()
```

```
Verify (Replace.Direction.GetItemCount (), 2)
Replace.Cancel.Click ()
```

Fuzzy verification

Fuzzy verification

There are situations when SilkTest cannot see the full contents of a control (such as a text field) because of the way that the application paints the control on the screen. For example, consider a text field whose contents are wider than the display area. In some situations (but not in others), the application clips the text to fit the display area before drawing it, meaning that SilkTest only sees the contents that are visible; not the entire contents.

Consequently, when you later do a `VerifyProperties` against this text field, it may fail inappropriately. For example, say the true contents of the text field are "29 Pagoda Street", but only "29 Pagoda" displays. Depending on exactly how the test is created and run, the expected value might be "29 Pagoda" whereas the value seen at runtime might be "29 Pagoda Street" (or vice versa). So the test would fail, even though it should pass.

To work around this problem, you can use fuzzy verification, where the rules for when two strings match are loosened. Using fuzzy verification, the expected and actual values do not have to exactly match. The two values are considered to match when one of them is the same as the first or last part of the other one. Specifically, VerifyProperties with fuzzy verification will pass whenever any of the following functions would return TRUE (where `actual` is the actual value and `expected` is the expected value):

- `MatchStr (actual + "*", expected)`
- `MatchStr ("*" + actual, expected)`
- `MatchStr (actual, expected + "*")`
- `MatchStr (actual, "*" + expected)`

(In string comparisons, * stands for any zero or more characters.)

For example, all the following would pass if fuzzy verification is enabled:

Actual Value	Expected Value
29 Pagoda	29 Pagoda Street
oda Street	29 Pagoda Street
29 Pagoda Street	29 Pagoda
29 Pagoda Street	oda Street

Enabling fuzzy verification

You enable fuzzy verification by using an optional second argument to VerifyProperties, which has this prototype:

```
VerifyProperties (WINPROPTREE WinPropTree [,FUZZYVERIFY FuzzyVerifyWhich])
```

where the `FUZZYVERIFY` data type is defined as:

```
type FUZZYVERIFY is BOOLEAN, DATACLASS, LIST OF DATACLASS
```

So, for the optional `FuzzyVerifyWhich` argument you can either specify `TRUE` or `FALSE`, one class name, or a list of class names.

FuzzyVerifyWhich value

FALSE (default)	Fuzzy verification is disabled.
One class	Fuzzy verification is enabled for all objects of that class. Example window.VerifyProperties ({…},Table) enables fuzzy verification for all Tables in window (but no other object).
List of classes	Fuzzy verification is enabled for all objects of each listed class. Example window.VerifyProperties ({…}, {Table, TextField}) enables fuzzy verification for all Tables and text fields in window (but no other object).

TRUE	Fuzzy verification is enabled only for those objects whose `FuzzyVerifyProperties` member is `TRUE`.
	To set the `FuzzyVerifyProperties` member for an object, add the following line within the object's declaration:
	`FUZZYVERIFY FuzzyVerifyProperties = TRUE`
	Example: If in the application's include file, the `DeptDetails` table has its `FuzzyVerifyProperties` member set to `TRUE`:
	`window ChildWin EmpData`
	`. . .`
	`Table DeptDetails`
	`FUZZYVERIFY FuzzyVerifyProperties = TRUE`
	And the test has this line:
	`EmpData.VerifyProperties ({...}, TRUE)`
	Then fuzzy verification is enabled for the `DeptDetails` table (and other objects in `EmpData` that have `FuzzyVerifyProperties` set to `TRUE`), but no other object.

Fuzzy verification takes more time than standard verification, so only use it when necessary.

For more information, see the `VerifyProperties` method.

Defining your own verification properties

You can also define your own verification properties. For more information, see *Defining new verification properties* method.

Verifying that a window or control has disappeared

1. Click **Record** > **Testcase** to begin recording a testcase and drive your application to the state you want to verify. To record a verification statement in an existing testcase, choose **Record** > **Actions**.
2. When you are ready to record a verification statement, position the mouse cursor over the object you want to verify, and press `Ctrl+Alt`. SilkTest displays the **Verify Window** dialog over your application window.
3. Click the **Property** tab. SilkTest lists the properties for the selected window or control on the right.
4. Make sure that only the `Exists` property is selected for the window or control.

   If additional properties are selected, the verification will fail because the actual list of properties will differ from the expected list.
5. Change the value in the **Property Value** field from `TRUE` to `FALSE`.
6. Click **OK** to accept the `Exists` property for the selected window or control. SilkTest closes the **Verify Window** dialog and displays the **Record Status** window. The testcase will verify that the window or

control has the property value of FALSE, verifying that the object has disappeared. If not, SilkTest writes an error to the results file.

Data Driven Testcases

Data Driven Workflow

The **Data Driven Workflow** guides you through the process of creating a data driven test case.

You can turn this workflow on and off by choosing **Workflows** > **Data Driven**.

To create and execute a data-driven test case, click each icon in the workflow bar to perform the relevant procedures including:

- Select a test case to data drive
- Find and replace values in the new test case with links to the data source
- Run the data driven test case, optionally selecting the rows and tables you want to use
- View test results

Using the Data Driven Workflow

The **Data Driven Workflow** is a way to help you create data driven testcases that pull their data from databases. Previous versions of SilkTest let you organize your variables into records in order to make them easier to understand and find, but there was no standard record format. Now, SilkTest uses a well-defined record format. This means that if you want to run your existing data driven testcases through the **Data Driven Workflow**, you will have to convert your record to the new record format. You can continue to run your existing data driven testcases as you always have, outside the **Data Driven Workflow**.

Before you begin to create and run data driven testcases, you need to:

1. Record a standard testcase.
2. Set up or identify the existing data source with the information you want to use to run the test.
3. Configure your Data Source Name (DSN), if you are not using the default (Silk DDA Excel).

The **Select Data Source** dialog box allows you to choose either the Silk DDA Excel or the Segue DDA Excel data source. For new data driven testcases, choose the Silk DDA Excel data source. Choose the Segue DDA Excel data source for backward compatablity. This allows existing `.g.t` files that reference Segue DDA Excel to continue to run.

Then, you can start using the **Data Driven Workflow** to:

1. Select a testcase to data drive. SilkTest copies the selected testcase and creates a new data driven testcase by adding a `"DD_"` prefix to the original name of the testcase. SilkTest also writes other data driven information to the new or existing data driven script file (`.g.t` file).
2. Find and replace values in the new testcase with links to the data source.
3. Run the data driven testcase, optionally selecting the rows and tables you want to use.

Overview of Data-Driven Test Cases

A data-driven test case lets you store data combinations in a list of items and invoke the test case once for each item, passing the data to the test case as a parameter. You can think of a data-driven test case as a template for a class of test cases. The benefits are easy to see:

- Data-driven test cases reduce redundancy in a test plan.
- Writing a single test case for a group of similar test cases makes it easier for you to maintain scripts.
- Data-driven test cases are reusable; adding new tests only requires adding new data.

You can use the **Data Driven Workflow** to help you create data-driven test cases that use data stored in databases. Previous versions of SilkTest Classic explained how to store data in a test plan, how to type in the data at the time you are running the test case, and how to pass in data from an external file. While those techniques still work, we strongly recommend that you now use the **Data Driven Workflow** which generates much of the necessary code for you and guides you through the required steps.

Regardless of the technique you use, the basic process for creating a data-driven test case is:

1. Create a standard test case. It will be very helpful to have a good idea of what you are going to test and how to perform the verification.
2. Identify the data in the test case and the 4Test data types needed to store this data.
3. Modify the test case to use variables instead of hard data.
4. Modify the test case to specify input arguments to be used to pass in the data. Replace the hard coded data in the test case with variables.
5. Call the test case and pass in the data, using one of four different techniques:
    - Use a database and the **Data Driven Workflow** to run the test case, the preferred method.
    - Click **Run** > **Testcase** and type the data in the **Run Testcase** dialog box.
    - In a QA Organizer test plan, insert the data as an attribute to a test description.
    - If the data exists in an external file, write a function to read the file and use a `main()` function to run the test case.

Working with data driven testcases

Consider the following when you are working with data driven testcases:

- The 4Test Editor contains additional menu selections and toolbars for you to use.
- SilkTest can data drive only one testcase at a time.
- You cannot duplicate testcase names. Data driven testcases in the same script must have unique names.
- The Classic 4Test editor is not available with data driven testcases (in `.g.t` files).
- You cannot create data driven testcases from testcases in `.inc` files; you can only create data driven testcases from testcases in `.t` or `.g.t` files. However, you can open a project, add the `*.inc`, select the testcase from the testcase folder of the project, and then select data drive.
- When you data drive a `[use '<script>.t']` is added to the data driven testcase. This is the link to the `.t` file where the testcase originated. If you add a testcase from another script file then another use line pointing to that file is added. If the Script file is in the same directory as the `<script.g.t>` then no path is given otherwise the absolute path is added to the use line. If this path changes, it is up to you to correct the path; SilkTest will not automatically update the path.
- When you open a `.g.t` file using **File** > **Open**, SilkTest automatically loads the data source information for that file. If you are in a `.g.t` file and that file's data source is edited, click **Edit** > **Data Driven** > **Reload Database** to refresh the information from the data source.
- If you add a new data driven testcase to an existing `.g.t` file that is fully collapsed, SilkTest expands the previous testcase, but does not edit it.

Code automatically generated by SilkTest

When you create a data driven testcase, SilkTest verifies that the DSN configuration is correct by connecting to the database, generates the 4Test code describing the DSN, and writes that information into the data driven script.

Do not delete or change the information created by SilkTest. If you do, you may not be able to run your data driven testcase.

When you click **OK** on the **Specify Data Driven Testcase** dialog, SilkTest automatically writes the following information to the top of your data driven script file.

The information is delivered "rolled up" (collapsed); in order to see the details you need to click on the plus sign to expand the code:

```
[+] // *** DATA DRIVEN ASSISTANT Section (!! DO NOT REMOVE !!) ***
    The .inc files used by the original testcases, and the .t file indicating
where the testcase just came from, in this case from Usability.t:
[ ] use "datadrivetc.inc"
[ ] use "Usability.t"
    A reference to the DSN, specifying the connect string, including username
and password, for example:
[ ] // *** DSN ***
[ ] STRING gsDSNConnect = "DSN=SILK DDA Excel;DBQ=C:\ddatesting
\TestExcel.xls;UID=;PWD=;"
```

Each data driven testcase takes as a single argument a record consisting of a record for each table that is used in the testcase. The record definition is automatically generated as shown here:

[+] // testcase VerifyProductDetails (REC_DATALIST_VerifyProductDetails rdVpd) [ ] // Name: REC_<Testcase name>. Fields Types: Table record types. Field Names: Table record type with 'REC_' replaced by 'rec' [-] type REC_DATALIST_VerifyProductDetails is record [ ] REC_Products recProducts [ ] REC_Customers recCustomers [ ] REC_CreditCards recCreditCards

Each table record contains the column names in the same order as in the database. Spaces in table and column names are removed. Special characters such as $ are replaced by underscores.

```
[ ] // *** Global record for each Table ***
[ ]
[-] type REC_Products_ is record
 [ ] STRING Item //Item,
 [ ] REAL Index //Index,
 [ ] STRING Name //Name,
 [ ] REAL ItemNum //ItemNum,
 [ ] STRING Price //Price,
 [ ] STRING Desc //Desc,
 [ ] STRING Blurb //Blurb,
 [ ] REAL NumInStock //NumInStock,
 [ ] INTEGER QtyToOrder //QtyToOrder,
 [ ] INTEGER OnSale //OnSale,
```

SilkTest writes a sample record for each table. This is the data used if you opt to use sample data on the *Run Testcase dialog*. A value from the original testcase is inserted into the sample record, even if there are syntax errors when that column is first used to replace a value.

```
[ ] // *** Global record containing sample data for each table ***
[ ] // *** Used when running a testcase with 'Use Sample Data from Script'
checked ***
[ ]
[-] REC_Products_ grTest_Products_ = {...}
 [ ] NULL // Item
 [ ] NULL // Index
 [ ] NULL // Name
 [ ] NULL // ItemNum
 [ ] NULL // Price
 [ ] NULL // Desc
 [ ] NULL // Blurb
 [ ] NULL // NumInStock
 [ ] 2 // QtyToOrder
 [ ] NULL // OnSale
```

```
[ ]
[ ] // *** End of DATA DRIVEN ASSISTANT Section ***
```

Tips and tricks for data driven testcases

There are several things to know about working with data sources while you are creating data driven testcases. See also Formatting MS Excel worksheets for use as a data source, below.

- You must have an existing data source with tables and columns defined before you data drive a testcase. However, the data source does not need to contain rows of data. You cannot use the Data Driven Workflow to create data sources or databases.
- If you have a table in your data source that has a long name (greater than 25 characters), all of the name may not be visible in the **Find and Replace** resizable menu bar in the 4Test Editor. You may find it helpful to change the size of the menu bar to display more of your table name.
- You cannot change to a different data source once you have started to find and replace values in a script. If you do, you will have problems with prior replacements. If you want to change your data source, you should create a new data driven script file.
- If you are working with a data source that requires a user name and password, you can add the username and password to the connect string in the `.g.t` file. The first example below shows how SQL Server requires a userid and password. `[ ] STRING gsDSNConnect = "DSN=USER.SQL.DSN;UID=SA;PWD=sesame;"` where UID=<your user ID> ("SA" in the example above) and where `PWD=<your password>` ("sesame" in the example above). On the other hand, the example below shows how the Connect string for a MS Excel DSN does not require user IDs or passwords: `[ ] STRING gsDSNConnect = "DSN=Silk DDA Excel;DBQ=C:\TestExcel.xls;UID=;PWD="`
- You can choose to run with a sample record if the table is empty; however, this record is not inserted into the database. The sample record is created by SilkTest when it replaces values from the testcase by the table and columns in your database.
- Real numbers should be stored as valid 4Test Real numbers with format: `[-]ddd.ddd[e[-]ddd]`, even though databases such as MS Excel allow a wider range of formats – for example, currencies and fractions.
- There are no restrictions on how you name your tables and columns within your data source. SilkTest automatically removes spaces, and converts dollar signs and other special characters to underscores when it creates the sample record and writes other code to your data driven testcase. SilkTest handles MS Excel and MS Access table names without putting quotation marks around them. This means that your table and column names will look familiar when you go to find and replace values.
- If you encounter the error "ODBC Excel Driver numeric field overflow" while running a testcase, check the Excel workbook that you are using as your data source. You may have some columns that are defined as STRING columns but contain numeric values in some of the rows. If you have a column that you want to treat as numeric strings rather than as numbers, either format the column as 'Text' or begin the number strings with a single-quote character. For example: '1003 instead of: 1003
- If modifying data sources in an existing Excel data sheet, use the **remove column** option to delete any data to be removed, as simply deleting from the cell, using clear contents, or copy/pasting content will not register correctly with the DDS file in SilkTest and may lead to a data source mismatch error: `*** Error: Incompatible types -- Number of list elements exceeds number of fields.`

Formatting MS Excel worksheets for use as a data source

Use the 'General' format for the columns of your worksheets. Here are specific suggestions for column formats based on the intended data type of the column:

Intended data type of column	Excel column format

STRING	If the column contains only text, no numbers, dates or booleans, then apply the 'General' format. If the column contains text and numbers, then you can still apply the 'General' format if you begin the number strings with a single-quote character. For example: '1003 instead of: 1003. Otherwise, apply the 'Text' format.
INTEGER or REAL	‘General' or 'Number' format.
BOOLEAN	‘General' format. Use only the values TRUE and FALSE.
DATETIME	‘Custom' format: yyyy-mm-dd hh:mm:ss. That agrees with the ISO format used by SilkTest DATETIME values.

Testing an application with invalid data

This topic assumes that you are familiar with data driving testcases. See *Data driving a testcase (pre 6.0)* and the example of creating a data driven testcase for more information.

To thoroughly test an application feature, you need to test the feature with invalid as well as valid data.

For example, the sample Text Editor application displays a message box if a user specifies a search string in the Find dialog that doesn’t exist in the document. To account for this, you can create a data driven testcase, like the following, that verifies that the message box appears and has the correct message:

```
type SEARCHINFO is record
   STRING  sText      // Text to type in document window
   STRING  sPos       // Starting position of search
   STRING  sPattern   // String to look for
   BOOLEAN bCase      // Case-sensitive or not
   STRING  sDirection // Direction of search
   STRING  sExpected  // The expected match
   STRING  sMessage   // The expected message in message box

testcase FindInvalidData (SEARCHINFO Data)
   TextEditor.File.New.Pick ()
   DocumentWindow.Document.TypeKeys (Data.sText + Data.sPos)
   TextEditor.Search.Find.Pick ()
   Find.FindWhat.SetText (Data.sPattern)
   Find.CaseSensitive.SetState (Data.bCase)
   Find.Direction.Select (Data.sDirection)
   Find.FindNext.Click ()

   MessageBox.Message.VerifyValue (Data.sMessage)
   MessageBox.OK.Click ()

   Find.Cancel.Click ()
   TextEditor.File.Close.Pick ()
   MessageBox.No.Click ()
```

The `VerifyValue` method call in this testcase verifies that the message box contains the correct string. For example, the message should be `Cannot find Ca` if the user enters `Ca` into the **Find** dialog and the document editing area does not contain this string.

Enabling and Disabling Workflow Bars

Only one workflow bar can be enabled at a time.

To enable or disable a workflow bar, click **Workflows** and then select the workflow bar that you want to turn on or off; for example, **Workflows** > **Basic**.

You can select one of the following workflows:

Workflow	Description
Basic workflow (Classic Agent)	Guides you through the process of creating a test case for the Classic Agent. Using this workflow bar, you create a project, automatically enable and test extension settings, configure the recovery system, and record and run a test case.
Basic workflow (Open Agent)	Guides you through the process of creating a test case for the Open Agent. Using this workflow bar, you create a project, configure application settings, and record and run a test case.
Data Driven workflow	Guides you through the process of creating a data driven test case.

Setting up the Data Source

Data source for data driven testcases

When you installed SilkTest, the `SILK DDA EXCEL` DSN was copied to your installation computer. This is the default DSN that SilkTest uses. This DSN uses a MS Excel 8.0 driver and does not have a particular workbook (.xls file) associated with it.

The **Select Data Source** dialog box allows you to choose either the Silk DDA Excel or the Segue DDA Excel data source. For new data driven testcases, choose the Silk DDA Excel data source. Choose the Segue DDA Excel data source for backward compatablity. This allows existing `g.t` files that reference Segue DDA Excel to continue to run.

You do not have to use the default DSN. If you plan to use a different DSN, see *Configuring your DSN* for more information.

You may use any of the following types of data sources:

- text files and comma separated value files (*.txt and *.csv files)
- MS Excel
- MS SQL Server
- MS Access
- Oracle
- Sybase SQL Anywhere

For information about specific version numbers for these data sources, see the Release Notes by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

Configuring Your DSN

The default DSN for data driven testcases, "Silk DDA Excel", is created when SilkTest is installed on your computer. If you want to use this default you do not need to configure your DSN.

The Select Data Source dialog box allows you to choose either the Silk DDA Excel or the Segue DDA Excel data source. For new data driven testcases, choose the Silk DDA Excel data source. Choose the Segue DDA Excel data source for backward compatablity. This allows existing g.t files that reference Segue DDA Excel to continue to run.

The following instructions show how to configure a Windows 2000 machine to use a different DSN than the Silk DDA Excel default.

1. Choose **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.
2. On the ODBC Data Source Administrator, click either the **System DSN** tab or the **User DSN** tab, depending on whether you want to configure this DSN for one user or for every user on this machine.
3. Click **Add**.
4. On the **Create New Data Source** dialog, select the driver for the data source and click **Finish**. If you ever need to restore SilkTest's default DSN, select the driver for Microsoft Excel Driver (*.xls).

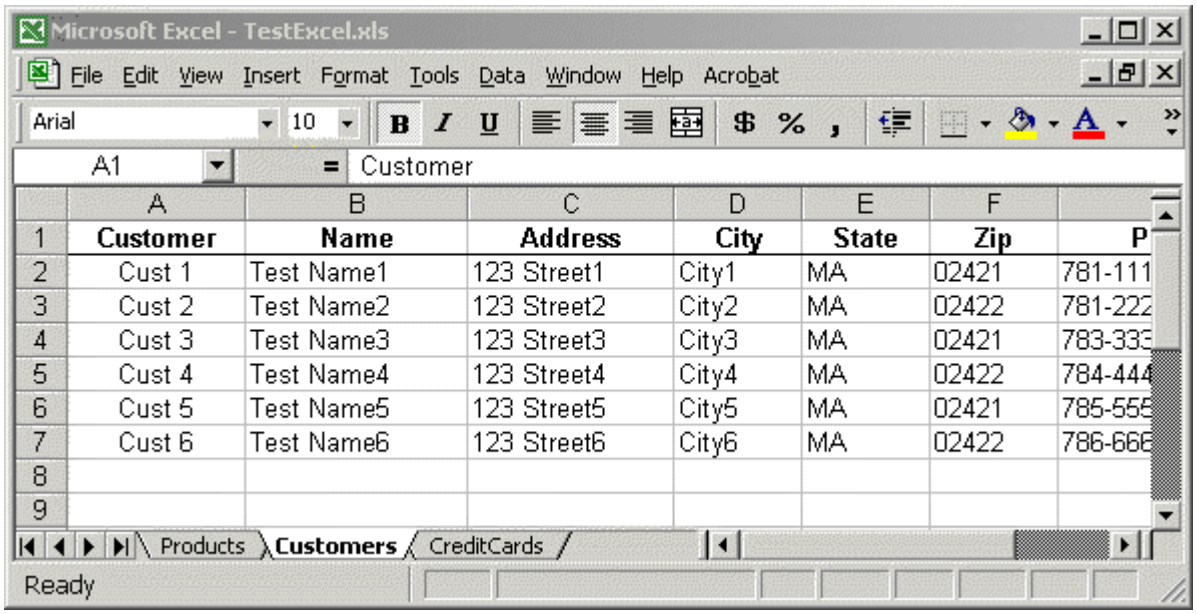
5. On the setup dialog, enter the name of the data source you selected . If you ever need to restore SilkTest's default, enter `Silk DDA Excel`. For more information about how to respond to the fields on this dialog, consult your database documentation or contact your database administrator.
6. Click **OK** to save your information.

Setting up a data source

Before you can run a data driven testcase you must set up a file that contains at least the tables(MS Excel calls these "worksheets") and columns that you want to use. The tables do not have to be populated with data, but you'll probably find it helpful to have at least one complete record filled out.

1. Open one of the data sources for data driven testcases, for example, MS Excel
2. Name at least one table (worksheet if you are using MS Excel) and create column names.
3. Save the file.

The MS Excel `TestExcel.xls` file shown below has three worksheets (tables) Products, Customers, and Credit Cards. The active worksheet, Customers has six columns: Customer, Name, Address, City, State, and Zip.



	A	B	C	D	E	F	
1	**Customer**	**Name**	**Address**	**City**	**State**	**Zip**	**P**
2	Cust 1	Test Name1	123 Street1	City1	MA	02421	781-111
3	Cust 2	Test Name2	123 Street2	City2	MA	02422	781-222
4	Cust 3	Test Name3	123 Street3	City3	MA	02421	783-333
5	Cust 4	Test Name4	123 Street4	City4	MA	02422	784-444
6	Cust 5	Test Name5	123 Street5	City5	MA	02421	785-555
7	Cust 6	Test Name6	123 Street6	City6	MA	02422	786-666
8							
9							

Using an Oracle DSN to Data Drive a Testcase

If you want to use an Oracle DSN to data drive a testcase, you must make some manual modifications after you select a testcase to data drive and after SilkTest generates code into the new testcase file:

1. Different schemas may contains tables with the same name. The table lists for the **Find/Replace Values** dialog and resizable menu bar and for the **Specify Rows** dialog will list the same table name once for each schema without indicating the schema. For each of those list items, the column list will contain the names of the columns in all of the tables with that name. You must know which columns are actually contained in your schema's table.
2. After finding and replacing values, you need to split each table record into separate records according to the schema. Do that for the sample record as well. The record names should have the form: `<Record prefix><schema>_<table>`. For example, if the schema is 'STUser' and the table is 'Customers' - The name of the table record type will be: REC_STUser_Customers. - The declaration for the field in the testcase record for the table will be: `REC_STUser_Customers recSTUser_Customers // Customers`.
3. You must run the testcase from a testplan, unless you are running all rows for all tables. You should use the **Specify Rows** dialog to build the ddatestdata value, then modify that value to include the

schema name in the query. Note that you must specify a query for every table, even if you want to run all rows for a table. To run all rows, leave the where clause blank.

Creating the Data Driven Testcase

Selecting a testcase to data drive

Before you select a testcase to data drive, there are several steps you need to complete. See *Using the Data Driven Workflow* for more information.

To select a testcase

To select a testcase for data driving while you are in a script, choose one of the following:

- Click **Tools**/**Data Drive Testcase**.
- Right-click and select **Data Drive Testcase**

Once you select a testcase, SilkTest copies the selected testcase and creates a new data driven testcase by adding a `DD_` prefix to the original name of the testcase. SilkTest also writes other data driven information to the new or existing data driven script file (`script.g.t`).

Finding and replacing values

Before you find and replace values in a testcase, there are several steps you need to complete. See *Using the Data Driven Workflow* for more information.

Values are text strings, numbers, and booleans (true/false) that exist in your original testcases. One of the steps in creating a data driven testcase is to find these values and replace them with references to columns in your data source.

SilkTest checks to make sure that each value you selected is appropriate for replacement by the column in your testcase. You can turn off this validation by clicking **Edit** > **Data Driven** > **Validate Replacements** while you are in a `.g.t` file. This means that the "Find" aspect of Find and Replace works as usual, but that the values you Replace are not validated. By turning off this checking, you suppress the error messages that SilkTest would have otherwise displayed. Any 4Test identifier or fragment of a string is considered an invalid value for replacement unless Validate Replacements is turned off.

If you are new to creating data driven testcases, we recommend that you keep this validation turned on.

To find and replace values

You find and replace values in a testcase using either the Find**/Replace Values** dialog or the **Find and Replace** resizable menu bar in the 4Test Editor. There are several ways to access the Find/Replace dialog.

- While the cursor is in a testcase of a `.g.t` file, right-click and select **Data Drive Testcase**. Specify the data source, data driven script, and data driven testcase. The **Find/Replace Values** dialog will appear automatically when the data driven script opens in the 4Test editor after you complete the **Specify Data Driven Testcase** dialog.
- After you have highlighted a value in a `.g.t` file, choose **Edit** > **Data Driven** > **Find** > **Replace Values**, or right-click and select **Find** > **Replace Values**

When you are using Find and Replace, sometimes a method requires a datatype that does not match the column you want to replace. For example, `SetText` requires a string, but you may want to set a number instead, or perhaps the database does not store data in the 4Test type that you would like to use. SilkTest can handle these kinds of conversions, with a few exceptions.

Running a data driven testcase

Once you have selected testcase to data drive, and found and replaced values, there are several ways to run the testcase.

Choose one of the following:

- Click **Run** > **Run** while in a `.g.t` file. This command runs `main()`, or if there is no `main()`, runs all testcases. For each testcase, this runs all rows for all tables used by the testcase.
- Click **Run** > **Testcase** and select the data driven testcase from the list of testcases on the Run Testcase dialog for all tables used by the testcase.
- Click **Run** > **Testcase** > **Run** to run the testcase for all rows for all tables used by the testcase.

Passing data to a testcase

Once you have defined your data driven testcase, you pass data to it, as follows:

- If you are not using the testplan editor, you pass data from a script's main function.
- If you are using the testplan editor, you embed the data in the testplan and the testplan editor passes the data when you run the testplan. See *Linking a testplan to a data driven testcase*.

Building queries

Before you define a query to access certain data in a data driven testcase, there are several steps you need to complete. See Using the Data Driven Workflow for more information.

Respond to the prompts on the Specify Rows dialog to create a query for a table. The following are examples of simple queries:

- To find and run the records of customers whose customer ID number is 1001: `(CUSTID = 1001)`.
- To find and run the records of customers whose names begin with the letters "F" or "G": `(CUST_NAME LIKE ‘F%’) OR (CUSTNAME LIKE ‘G%’)`.

See the description of the enter values area to see examples of more complex queries.

Adding a Data Driven Testcase to a Testplan

You can run a data driven testcase from a testplan as either a data driven testcase or as a regular testcase. To distinguish between the two cases, there are two keywords for you to use:

- `ddatestcase` specifies the name of a testcase that runs as a data driven testcase
- `ddatestdata` specifies the list of rows that will be run with the data driven testcase

If the testcase is specified with the keyword `ddatestcase`, it is run as a data driven testcase. Use this keyword only with data driven testcases.

To specify a data driven testcase in a testplan

- Add keyword `ddatestcase` in front of the testcase name
- Add the keyword `ddatestdata` as a list of queries that specify the particular rows you want the testcase to run with. The list of queries is represented as a single `LIST OF STRING` parameter.

Rules for using data driven keywords

- The `ddatestdata` keyword requires simple select queries. To specify the row you want to run a testcase with, use the `ddatestdata` keyword with the format: `select * from <table> where ....`
- The keyword `ddatestcase` cannot be a level above the script file and still work. The script file has to be at the same level or above it.

- A testplan needs to specify a testcase using either the keyword testcase or the keyword `ddatestcase`. Using both causes a compiler error.
- If the `ddatestdata` keyword is present, then the `ddatestcase` is run using the `ddatestdata` value as the rows to run.
- The default is to run all rows for all tables. The value for `ddatestdata` for this is `ALL_ROWS_FOR_ALL_TABLES`.
- Using the keyword testdata in a test item with keyword `ddatestcase` will cause a compiler error.
- If the testcase is specified with the keyword testcase, then the testcase is run as a regular testcase and the testdata keyword or symbols must be present to specify the value that will be passed as the regular argument. This value must be a record of the type defined for the `ddatestcase`, in other words of type `REC_DATALIST_<Testcase name>`.

You can add a data driven testcase to a testplan by using the **Testplan Detail** dialog or by editing the testplan directly. However, if you edit the testplan directly, then the keywords are not automatically validated and it is your responsibility to make sure that the keywords (testcase versus `ddatestcase`; testdata versus `ddatestdata`) are appropriate for the intended execution of the testcase.

Whenever you use the **Test Detail** dialog, be sure to click the Testcases button and select the testcase from the list. That will ensure that the proper keywords are inserted into the testplan.

Using sample records data within testplans

To run a testcase with the sample record within a testplan, you must manually input the testdata, in the format `ddatestdata: {"USE_SAMPLE_RECORD_<tablename>"}`

For example:

```
script: example.t
ddatestcase: sampletc
ddatestdata: {"USE_SAMPLE_RECORD_SpaceTable$"}
```

You must put the `USE_SAMPLE_RECORD_` prefix in front of each table name that you want to run against. If you are using two tables, you need to input the prefix twice, as shown below with two tables named "Table1" and "Table2":

```
ddatestdata: {"USE_SAMPLE_RECORD_Table1","USE_SAMPLE_RECORD_Table2"}
```

Using a main function in the script

Although most of the script files you create contain only testcases, in some instances you need to add a function named main to your script. You can use the main function to pass data to testcases as well as control the order in which the testcases in the script are executed.

When you run a script file using Run/Run:

- If the script file contains a main function, the main function is executed, then execution stops. Only testcases and functions called by main will be executed, in the order in which they are specified in main.
- If the script does not contain a main function, the testcases are executed from top to bottom.

Example

The following template shows the structure of a script that contains a main function that passes data to a data driven testcase:

```
main ()
// 1. Declare a variable to hold current record
// 2. Store all data for testcase in a list of records
// 3. Call the testcase once for each record in the list
```

Using this structure, the following example shows how to create a script that defines data records and then calls the sample testcase once for each record in the list:

```
type SEARCHINFO is record
   STRING  sText      // Text to type in document window
```

```
   STRING  sPos       // Starting position of search
   STRING  sPattern   // String to look for
   BOOLEAN bCase      // Case-sensitive or not
   STRING  sDirection // Direction of search
   STRING  sExpected  // The expected match

main ()
   SEARCHINFO Data
   list of SEARCHINFO lsData = {...}
      {"Test Case", "<END>", "C", TRUE, "Up", "C"}
      {"Test Case", "<END>", "Ca", TRUE, "Up", "Ca"}
      // additional data records can be added here
   for each Data in lsData
      FindTest (Data)

testcase FindTest (SEARCHINFO Data)
   TextEditor.File.New.Pick ()
   DocumentWindow.Document.TypeKeys (Data.sText + Data.sPos)
   TextEditor.Search.Find.Pick ()
   Find.FindWhat.SetText (Data.sPattern)
   Find.CaseSensitive.SetState (Data.bCase)
   Find.Direction.Select (Data.sDirection)
   Find.FindNext.Click ()
   Find.Cancel.Click ()
   DocumentWindow.Document.VerifySelText ({Data.sExpected})
   TextEditor.File.Close.Pick ()
     MessageBox.No.Click ()
```

When you select **Run** > **Run**, the main function is called and the `FindTest` testcase will be executed once for every instance of Data in lsData (the list of SEARCHINFO records). In the script shown above, the testcase will be run twice. Here is the results file that is produced:

```
Script findtest.t - Passed
Passed:  2 tests (100%)
Failed:  0 tests (0%)
Totals:  2 tests, 0 errors, 0 warnings

Testcase FindTest ({"Test Case", "<END>", "C", TRUE, "Up", "C"}) - Passed
Testcase FindTest ({"Test Case", "<END>", "Ca", TRUE, "Up", "Ca"}) - Passed
```

Notice that with data driven testcases, SilkTest records in the results file the parameters that are passed in.

In this sample data driven testcase, the testcase data is stored in a list within the script itself. It is also possible to store the data externally and read records into a list using the FileReadValue function.

Using do...except to handle an exception

The `VerifyValue` method, like all 4Test verification methods, raises an exception if the actual value does not match the expected (baseline) value. When this happens, SilkTest halts the execution of the testcase and transfers control to the recovery system. The recovery system then returns the application to the base state. See the recovery system.

However, suppose you don't want SilkTest to transfer control to the recovery system, but instead want to trap the exception and handle it yourself. For example, you might want to log the error and continue executing the testcase. To do this, you can use the 4Test `do...except` statement and related statements, which allow you to handle the exception yourself.

Property Sets

Verifying properties as sets

To make your testing easier, properties are organized into sets. A property set consists of a list of properties and the class associated with each property. A number of property sets is predefined for your convenience.

Properties and attributes in this context are similar—they both are used to verify a characteristic of an object. However, properties are more encompassing, more flexible, and easier to use. For example, using attributes you can only verify one attribute at a time (or verify every attribute for an object and all its children); using properties you can verify selected properties of an object and any or all of its children at the same time.

All property sets reside in the file named in the Data File for Property Sets field in the **Recorder Options** dialog. The default file location is your SilkTest installation directory. To make sure that all testers in your group have access to the same property sets file, place the file on a shared drive and specify the full path in the **Data File for Property Sets** field.

If you selected enhanced support for Visual Basic, your property set file is `vbprpset.ini`. If you did not select enhanced support for VB, then your property set file is `propset.ini`.

You can configure property sets to suit your needs, even combining frequently used property sets into a new larger property set.

Creating a new property set

1. Click **Options** > **Property Sets**.
   SilkTest displays the **Property Sets** dialog, which lists all existing property sets. When you are working with the SilkTest Classic Agent, you can also click **Define** on the **Verify Window** dialog.
2. On the **Property Sets** dialog, click **New**.
3. Specify a name for the new property set in the **Name** field. Property set names are not case sensitive. They can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
4. Specify a class in the **Class** field and then a property of that class in the **Property** field.
5. Click **Add**.
   SilkTest adds the class-property pair is added to the list box. The class or property name is not validated here, so type carefully. Invalid names are ignored at runtime. If you make a mistake, select the class-property pair and click **Edit**.
6. Repeat steps 4 and 5 for as many class-property pairs as you want to add. Delete any class-property pairs you don’t want to include by selecting them and clicking **Remove**.
7. Once the list of classes and properties is correct, click **OK**.
   SilkTest closes the **New Property Set** dialog and displays the new property set in the **Property Sets** list box.
8. Click **Close**.

Combining two property sets

1. Click **Options** > **Property Sets** to display the Property Sets dialog. When you are working with the SilkTest Classic Agent, you can also click Define on the Verify Window dialog.
2. Click the **Combine** button.
3. On the **Combine Property Sets** dialog, specify a name for the new property set in the **Name** field.
4. Select at least two property sets from the Property sets to combine list box and click **OK**. SilkTest closes the **Combine Property Sets** dialog and displays the new property in the **Property Sets** list box, along with the constituent sets.

If you modify any of the constituent sets, the combined set will be modified as well.

Deleting a property set

1. Click **Options** > **Property Sets**. When you are working with the SilkTest Classic Agent, you can also click **Define** on the **Verify Window** dialog.
2. Select the name of the property set you want to delete from the **Property Sets** list box and then click **Remove**.
3. SilkTest prompts you are to confirm the deletion. Click **Yes** to delete the property set.
4. Click **Close**.

Editing an existing property set

1. Choose **Options** > **Property Sets** to display the **Property Sets** dialog. When you are working with the SilkTest Classic Agent, you can also click **Define** on the **Verify Window** dialog.
2. On the **Property Sets** dialog, select a property set from the **Property Sets** list box and click **Edit**.
3. Take any of the following actions:

Edit the property set name	Edit the name in the Name field.
Add a class-property pair	1. Specify a class in the **Class** field. 2. Specify a property for the class in the **Property** field. 3. Click **Add**.
Delete a class-property pair	Select a class-property pair and click Remove. The pair is deleted from the list box.
Edit a class-property pair	1. Select a class-property pair and click **Edit**. The class and property appear in the text fields at the bottom of the dialog and the **Add** pushbutton becomes **Replace**. 2. Modify the class, property, or both, and click **Replace**. SilkTest displays the class-property pair in the list box.

4. When you finish editing, click OK.
5. Click OK to close the **Property Set** dialog.

Specifying a class-property pair

There are several ways to specify class-property pairs.

- They can be specified as a full class name and a full property name. For example, to specify the **State** property for the `CheckBox` class, enter **CheckBox** in the **Class** field and **State** in the **Property** field.
- You can use the * wildcard character for partial matches. The asterisk matches zero or more characters. For example, specifying * as a class name matches all classes. Specifying Text* as a class name matches all classes that begin with the string "Text".
- You can apply the rule of inheritance to property sets; that is, the properties of a class are inherited by its child classes. For example, specifying the **Enabled** property and the `Control` class as a pair means that the **Enabled** property of all classes descended from `Control` are also implicitly included in the property set.

Testing in Your Environment

Testing Adobe Flex Applications

Overview of Adobe Flex Support

SilkTest Classic provides built-in support for testing Adobe Flex (Flex) applications using Internet Explorer, Firefox, or the Standalone Flash Player, and Adobe AIR applications built with Flex 4 or later.

SilkTest Classic also supports multiple application domains in Flex 3.x and 4.x applications, which enables you to test sub-applications. SilkTest Classic recognizes each sub-application in the locator hierarchy tree as an application tree with the relevant application domain context. At the root level in the locator attribute table, Flex 4.x sub-applications use the `SparkApplication` class. Flex 3.x sub-applications use the `FlexApplication` class.

For information on the supported versions and potential known issues, refer to the *Release Notes*.

Sample Applications

To access the SilkTest Classic sample Flex applications, go to *http://demo.borland.com/flex/SilkTest2011/index.html*.

Object Recognition

Flex applications support hierarchical object recognition and dynamic 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.

When you record a test case with the Open Agent, SilkTest Classic creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Existing Flex test cases that use hierarchical object recognition or dynamic object recognition without locator keywords in an INC file are supported. You can replay these tests, but you cannot record new tests with hierarchical object recognition or dynamic object recognition without locator keywords in an INC file. However, you can manually create tests as needed. Then, replay the tests at your convenience. For instance, any test cases that you recorded with SilkTest 2008 use hierarchical object recognition. You can replay these tests in SilkTest Classic.

Supported Controls

For a complete list of the record and replay controls available for Flex testing, refer to the *Flex Class Reference* in the *4Test Language* section of the Help.

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

Agent Support

When you create a SilkTest Classic Flex project, the Open Agent is assigned as the default Agent.

Configuring Flex Applications to Run in Adobe Flash Player

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

- The developer who creates the Flex application must compile the application as an EXE file. When a user launches the application, it will open in Flash Player. Install Windows Flash Player from *http://www.adobe.com/support/flashplayer/downloads.html*.
- The user must have Windows Flash Player Projector installed. When a user opens a Flex .SWF file, he can configure it to open in Flash Player. Windows Flash Projector is not installed when Flash Player is installed unless you install the Adobe Flex developer suite. Install Windows Flash Projector from *http://www.adobe.com/support/flashplayer/downloads.html*.

To configure Flex applications to run in Flash Player:

1. If your operating system is Windows 7 or Windows 2008 R2, you need to perform the following steps to configure Flash Player to run as administrator:
   a) Right-click the Adobe Flash Player program shortcut or the `FlashPlayer.exe` file, then click **Properties**.
   b) In the **Properties** dialog box, click the **Compatibility** tab.
   c) Check the **Run this program as an administrator** check box and then click **OK**.
2. Start the .SWF file in Flash Player from the command prompt (cmd.exe) by typing `"<Application_Install_Directory>\<ApplicationName>.swf"`

   For example, to start the SilkTest sample application, `Explorer.swf`, in Flash Player, type `"<SilkTest_Install_Directory>\ng\sampleapplications\Flex\<flex version number>\FlexControlExplorer<flex version number>_withAutomation\Explorer.swf"`. By default, the `<SilkTest_Install_Directory>` is located at `Program Files\Silk\SilkTest`.

Configuring Flex Applications for Adobe Flash Player Security Restrictions

The security model in Adobe Flash Player 10 has changed from earlier versions. When you record tests that use Adobe Flash Player (Flash Player), recording works as expected. However, when you play back tests, unexpected results occur when high-level clicks are used in certain situations. For instance, a file reference dialog box cannot be opened programmatically and when you attempt to play back this scenario, the test fails because of security restrictions.

To work around the security restrictions, you can perform a low-level click on the button that opens the dialog box. To create a low-level click, add a parameter to the `Click` method.

For example, instead of using `SparkButton::Click()`, use `SparkButton::Click(1)`. A `Click()` without parameters is a high-level click and a click with parameters, such as the button, is replayed as a low-level click.

1. Record the steps that use Flash Player.
2. Navigate to the `Click` method and add a parameter.

   For example, to open the **Open File** dialog box, specify `SparkButton("@caption='Open File Dialog…').Click(1)`.

   When you play back the test, it works as expected.

Testing Adobe Flex Applications

SilkTest provides built-in support for testing Adobe Flex applications. SilkTest also provides several sample Adobe Flex test applications. To access the samples, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Adobe Flex/Flex Sample Applications** and choose the sample application you want to use.

For up-to-date information about supported versions, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes** to view the *Release Notes*.

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

- Enable your Flex Application for testing.
- Create testable Flex Applications.
- Code Flex containers.
- Implement automation support for the custom controls.

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

1. Verify that the prerequisites for testing Adobe Flex applications are met.
2. Create a new project.
3. Configure the Web Application.
4. Record the testcase.
5. Run the testcase.
6. Customize the Adobe Flex scripts.

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

Customizing Adobe Flex Scripts

You can manually customize your Flex scripts. You can insert verifications using the **Verification** wizard. Or, you can insert verifications manually using the `Verify` function on Flex object properties.

To customize Adobe Flex scripts:

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

   For example, the following code adds a verification call to your script:

```
Desktop.Find("//BrowserApplication").Find("//BrowserWindow")
.Find("//FlexApplication[@caption='explorer']").Find("//
FlexButton[@caption='OK']")
.VerifyProperties({...})
```

Each Flex object has a list of properties that you can verify. For a list of the properties available for verification, review the `Flex.inc` file. To access the file, navigate to the `<SilkTest directory>\extend\Flex` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\Flex\Flex.inc`.

Styles in Adobe Flex Applications

For applications developed in Adobe Flex 3.x, SilkTest does not distinguish between styles and properties. As a result, styles are exposed as properties. However, with Adobe Flex 4.x, all new Flex controls, which are prefixed with Spark, such as `SparkButton`, do not expose styles as properties. As a result, the

`GetProperty()` and `GetPropertyList()` methods for Flex 4.x controls do not return styles, such as color or fontSize, but only properties, such as text and name.

The `GetStyle`(string *styleName*) method returns values of styles as a string. To find out which styles exist, refer to the Adobe Help located at *http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/package-detail.html*.

If the style is not set, a StyleNotSetException occurs during playback.

For the Flex 3.x controls, such as `FlexTree`, you can use `GetProperty()` to retrieve styles. Or, you can use `GetStyle()`. Both the `GetProperty()` and `GetStyle()` methods work with Flex 3.x controls.

Calculating the Color Style

In Flex, the color is represented as a number. It can be calculated using the following formula:

red*65536 + green*256 + blue

Example

In this example, the `GetProperty()` and `GetStyle()` methods are used to retrieve styles:

```
Window myTree = Application.Find("//
FlexTree[@caption='myTree']")
COLOR  c = {170, 179, 179}
Verify(myTree.DisabledColor, c)
Verify(myTree.GetProperty("disabledColor"), {170, 179, 179})
Verify(myTree.GetStyle("disabledColor"), "11187123")
```

The number `11187123` for the color calculates as `170*65536 + 179*256 + 179`.

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.

The attributes that SilkTest supports for Flex applications include:

`automationName`	The name of the application.
`caption`	Similar to `automationName`.
`automationClassName`	For example `FlexButton`.
`className`	The fully qualified name of the implementation class, for example `mx.controls.Button`.
`automationIndex`	The index of the control in the view of the `FlexAutomation`, for example `index:1`.
`index`	Similar to `automationIndex` but without the prefix, for example `1`.
`id`	The identifier of the control.
`windowId`	Similar to `id`.
`label`	The label of the control.

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

Dynamically Invoking Adobe Flex Methods

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

Call dynamic methods on objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a control, use the `GetDynamicMethodList`() method.

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty`() method. To retrieve a list of supported dynamic properties for a control, use the `GetPropertyList`() method.

Note: Most properties are read-only and cannot be set.

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

All built-in SilkTest types	SilkTest types includes primitive types, such as boolean, int, and string, lists, and other types, such as Point.

Returned Values

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

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

Example

A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

```
customControl.Invoke("Reset")
REAL sum = customControl.DynamicInvoke("Add",{1,2})
```

Testing Multiple Flex Applications on the Same Web Page

When multiple Flex applications exist on the same Web page, SilkTest uses the Flex application ID or the application size property to determine which application to test. If multiple applications exist on the same page, but they are different sizes, SilkTest uses the size property to determine on which application to perform any actions. SilkTest uses JavaScript to find the Flex application ID to determine on which application to perform any actions if:

- multiple Flex applications exist on a single Web page.
- those applications are the same size.

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

To test multiple Flex applications that are different sizes on a single Web page , follow the steps in *Testing Adobe Flex Applications*.

To test multiple Flex applications that are the same size on a single Web page, perform the following steps:

1. Enable JavaScript.
   - In Internet Explorer:
     1. Choose **Tools** > **Internet Options**.
     2. Click the **Security** tab.
     3. Click **Custom level**.
     4. In the **Scripting** section, under **Active Scripting**, click **Enable** and click **OK**.
   - In Mozilla Firefox:
     1. Choose **Tools** > **Options**.
     2. Click **Content** and then check the **Enable JavaScript** check box.
     3. Click **OK**.
2. Follow the steps in *Testing Adobe Flex Applications*.

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

SilkTest provides sample applications that demonstrate multiple applications on a single Web page. You must download the Flex sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, to access the samples, choose `Start/Programs/Silk/SilkTest <version>/Sample Applications/Adobe Flex/Flex Sample Applications` and choose the sample application you want to use.

Adobe AIR Support

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

SilkTest provides a sample Adobe AIR application. You can access the sample application at *http://demo.borland.com/flex/SilkTest2011/index.html* and then click the Adobe AIR application that you want to use. You can select the application with or without automation. In order to execute the AIR application, you must install the Adobe AIR Runtime.

Adobe Flex Exception Values

Exception values are generated under given error conditions. Flex support defines the following set of exception values:

`E_FLEX_REPLAY`	A generic exception, which is thrown when no other known exception occurs in Flex.
`E_FLEX_REPLAY_EVENT`	An error occurred when replaying the Flex event.
`E_FLEX_REPLAY_METHOD`	An error occurred when replaying the Flex method.
`E_FLEX_REPLAY_READ_PROPERTY`	An error occurred when reading a property.
`E_FLEX_REPLAY_WRITE_PROPERTY`	An error occurred when writing a property.
`E_FLEX_REPLAY_STYLE_NOT_SET`	The style is not set to a Flex object.

E_FLEX_REPLAY_SUPPORTS_TABLUAR	The property used is meant for use with tabular data. However, the specified class does not support tabular data.
E_FLEX_REPLAY_INVALID_FLEX_SDK_VERSION	If you replay a Flex 3.x event, method, or property in a Flex 2.0 environment, this error occurs.
E_VO_PROPERTY_NOT_FOUND	When reading or writing a property, if the property is not defined for the object, this exception occurs.

The `E_VO_PROPERTY_NOT_FOUND` exception can also be thrown when you test Flex, but it is not limited to the Flex environment.

Overview of the Flex Select Method Using Name or Index

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

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

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

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

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

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

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

Selecting an Item in the FlexDataGrid Control

You can select an item in the FlexDataGrid control using the following procedures.

If you know the index value of the FlexDataGrid item, use the `SelectIndex` method.

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

1. If you know the content value of the FlexDataGrid item, use the `Select` method.

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

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

The following example selects an item using the `Select` method (randomly).

```
[ ] LIST OF LIST OF STRING  allVisibleItems
[ ] window dataGrid =
AdobeFlashPlayer9.FlexApplication0.Index0.Index1.SwfLoader.ControlsSimpleDataG
ridSwf.DataGridControlExample.Dg
[ ]
[ ] // lets get all currently visible items
[ ] allVisibleItems = dataGrid.GetValues(dataGrid.firstVisibleRow,
dataGrid.lastVisibleRow)
[ ]
[ ] // pick a random element that we want to select
[ ] integer randomRow = RandInt(dataGrid.firstVisibleRow,
dataGrid.lastVisibleRow)
[ ] LIST OF STRING randomRowItems = allVisibleItems[randomRow]
[ ] print("This is the row we want to select: {randomRow}")
[ ]
[ ] // now lets construct the string we need for the select method
[ ] STRING selectString
[ ] STRING itemText
[ ] INTEGER col = 0
[-] for each itemText in randomRowItems
[-] if col == 0
[ ] selectString = "*{itemText}*"
[-] else
[ ] selectString = selectString + " | {itemText}"
[ ] col++
[ ]
[ ] // now lets select the item
[ ] print("We will select {selectString}")
[ ] dataGrid.Select(selectString)
```

Enabling Your Flex Application for Testing

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

Adobe Flex Automation Package

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

Automation libraries	The `automation.swc` and `automation_agent.swc` libraries are the implementations of the delegates for the Flex framework components. The `automation_agent.swc` file and its associated resource bundle are the generic agent mechanism. An agent, such as the SilkTest Agent, builds on top of these libraries.
Samples	The SilkTest Flex Automation SDK is based on the Automation API for Flex. The SilkTest Automation SDK supports the same components in the same manner that the Automation API for Flex supports them. For instance, the typekey statement in the Flex Automation API does not support all keys. You can use the input text statement to resolve this issue. For more information about using the Flex Automation API, refer to the *Adobe Flex Release Notes*.

SilkTest Automation Package

SilkTest's Open Agent uses the Adobe Flex automation agent libraries. The `FlexTechDomain.swc` file contains the SilkTest specific implementation.

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

- *Linking Automation Packages to Your Flex Application*
- *Run-Time Loading*

Linking Automation Packages to Your Flex Application

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

When you embed functional testing classes in your application SWF file at compile time, the size of the SWF file increases. If the size of the SWF file is not important, use the same SWF file for functional testing and deployment. If the size of the SWF file is important, generate two SWF files, one with functional testing classes embedded and one without. Use the SWF file that does not include the embedded testing classes for deployment.

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

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

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

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

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

Precompiling the Flex Application for Testing

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

To precompile the Flex application for testing:

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

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

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

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

   The configuration files are located at:

Configuration File	Location
Adobe Flex 2 SDK	`<flex_installation_directory>/frameworks/flex-config.xml`
Adobe Flex Data Services	`<flex_installation_directory>/flex/WEB-INF/flex/flex-config.xml`

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

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

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

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

Run-Time Loading

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

To use the Flex Automation Launcher for run-time loading:

1. Copy the content of the `Silk\SilkTest\ng\AutomationSDK\Flex\<VERSION>\FlexAutomationLauncher` directory into the directory of the Flex application that you are testing.
2. Open `FlexAutomationLauncher.html` in Windows Explorer and add the parameter `?automationurl=YourApplication.swf` as a suffix to the file path.

   Note: `YourApplication.swf` is the name of the .swf file for your Flex application.

3. Add `file:///` as a prefix to the file path.

For example, if your file URL includes a parameter, such as `?automationurl=explorer.swf`, type `file:///C:/Program%20Files/Silk/SilkTest/ng/sampleapplications/Flex/3.2/FlexControlExplorer32/FlexAutomationLauncher.html?automationurl=explorer.swf`.

Run-time loading includes the following limitations:

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

```
import flash.display.*;
import flash.net.URLRequest;
import flash.system.ApplicationDomain;
import flash.system.LoaderContext;
var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("RuntimeClasses.swf");
var context:LoaderContext = new LoaderContext();
context.applicationDomain = ApplicationDomain.currentDomain;
loader.load(request, context);
```

Using the Command Line to Add Configuration Information

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

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

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

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

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

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

Passing Parameters into a Flex Application

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

To see a sample of both methods, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Adobe Flex** > **Flex Sample Applications**, type a value in the **Parameter Sample** field, and then click the links.

Creating Testable Flex Applications

As a Flex developer, you can employ techniques to make Flex applications easier to test. These techniques include:

Providing Meaningful Identification of Objects

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

To provide meaningful identification of objects:

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

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

Avoiding Duplication of Objects

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

To avoid duplicating objects:

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

Flex AutomationName and AutomationIndex Properties

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

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

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

Setting the Flex automationName Property

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

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

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

Example

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

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

Example

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

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

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

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

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

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

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

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

Setting the Flex Select Method to Use Name or Index

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

To record Flex `Select` methods to use the `Index` of the control that you select:

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

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

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

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

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

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

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

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

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

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

   With this adaption the IndexBasedSelection is used for recording FlexList::SelectIndex events.

   Setting the `EnableIndexBasedSelection=` to `false` in the code or removing the Boolean returns recording to using the name (FlexList::Select events).

5. Re-start your Flex application and the Open Agent, in order for these changes to become active.

Coding Flex Containers

Containers differ from other kinds of controls because they are used both to record user interactions, such as when a user moves to the next pane in an Accordion container, and to provide unique locations for controls in the testing scripts.

Adding and Removing Containers from the Automation Hierarchy

The automated testing feature reduces the amount of detail about nested containers in the included scripts. It removes containers that have no impact on the results of the test or on the identification of the controls from the script. This applies to containers that are used exclusively for layout, such as the HBox, VBox, and Canvas containers, except when they are being used in multiple-view navigator containers, such as the ViewStack, TabNavigator, or Accordion containers. In these cases, they are added to the automation hierarchy to provide navigation.

Many composite components use containers, such as Canvas or VBox, to organize their children. These containers do not have any visible impact on the application. As a result, you typically exclude these containers from testing because there is no user interaction and no visual need for their operations to be recordable. By excluding a container from testing, the related test script is less cluttered and easier to read.

To exclude a container from being recorded, without excluding the children of the container, set the `showInAutomationHierarchy` property of the container to false. This property is defined by the `UIComponent` class, so all containers that are a subclass of `UIComponent` have this property. Children of containers that are not visible in the hierarchy appear as children of the next highest visible parent.

The default value of the `showInAutomationHierarchy` property depends on the type of container. For containers such as Panel, Accordion, Application, DividedBox, and Form, the default value is *true*; for other containers, such as Canvas, HBox, VBox, and FormItem, the default value is *false*.

The following example forces the VBox containers to be included in the hierarchy of the test script:

```
<?xml version="1.0"?>
<!-- at/NestedButton.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Panel title="ComboBox Control Example">
<mx:HBox id="hb">
<mx:VBox id="vb1" showInAutomationHierarchy="true">
<mx:Canvas id="c1">
<mx:Button id="b1" automationName="Nested Button 1" label="Click Me" />
</mx:Canvas>
</mx:VBox>
<mx:VBox id="vb2" showInAutomationHierarchy="true">
<mx:Canvas id="c2">
<mx:Button id="b2" automationName="Nested Button 2" label="Click Me 2" />
</mx:Canvas>
```

```
</mx:VBox>
</mx:HBox>
</mx:Panel>
</mx:Application>
```

Working with Multiview Containers

Avoid using the same label on multiple tabs in multiview containers, such as the TabNavigator and Accordion containers. Although it is possible to use the same labels, this is generally not an acceptable UI design practice and can cause problems with control identification in your testing environment.

Flex Automation Testing Workflow Within SilkTest

The SilkTest workflow for testing Flex applications includes:

- Initialization
- Recording
- Playback

Flex Automated Testing Initialization

When the user launches the Flex application, the following initialization events occur:

1. The automation initialization code associates component delegate classes with component classes.
2. The component delegate classes implement the `IAutomationObject` interface.
3. An instance for the `AutomationManager` is created in the mixin `init`() method. The `AutomationManager` is a mixin.
4. The `SystemManager` initializes the application. Component instances and their corresponding delegate instances are created. Delegate instances add event listeners for events of interest.
5. The `FlexTechDomain` of SilkTest is a mixin. In the `FlexTechDomain` `init`() method, the `FlexTechDomain` registers for the `SystemManager`.APPLICATION_COMPLETE event. When the event is received, it creates a `FlexTechDomain` instance.
6. The `FlexTechDomain` instance connects by using a TCP/IP socket to the SilkTest Agent on the same machine that registers for record/replay functionality.
7. The `FlexTechDomain` requests information about the automation environment. This information is stored in XML files and is forwarded from the SilkTest Agent to the `FlexTechDomain`.

Flex Automated Testing Recording

When the user records a new testcase in SilkTest for a Flex application, the following events occur:

1. SilkTest calls the SilkTest Agent to start recording. The Agent forwards this command to the `FlexTechDomain` instance.
2. `FlexTechDomain` notifies `AutomationManager` to start recording by calling `beginRecording`(). The `AutomationManager` adds a listener for the AutomationRecordEvent.RECORD event from the SystemManager.
3. The user interacts with the application. For example, the user clicks a `Button` control.
4. The `ButtonDelegate.clickEventHandler`() method dispatches an AutomationRecordEvent event with the click event and `Button` instance as properties.
5. The `AutomationManager` record event handler determines which properties of the click event to store based on the XML environment information. It converts the values into proper type or format and dispatches the record event.
6. The FlexTechDomain event handler receives the event. It calls the `AutomationManager.createID`() method to create the `AutomationID` object of the button. This object provides a structure for object identification. The `AutomationID` structure is an array of AutomationIDParts. An AutomationIDPart is created by using `IAutomationObject`. The `UIComponent.id`, `automationName`, `automationValue`, `childIndex`, and `label` properties of the `Button` control are read and stored in

the object. The label property is used because the XML information specifies that this property can be used for identification of the `Button`.

7. `FlexTechDomain` uses the `AutomationManager.getParent()` method to get the logical parent of `Button`. The AutomationIDPart objects of parent controls are collected at each level up to the application level.
8. All the AutomationIDParts are included as part of the `AutomationID` object.
9. The `FlexTechDomain` sends the information in a call to SilkTest.
10. When the user stops recording, the `FlexTechDomain.endRecording()` method is called.

Flex Automated Testing Playback

When the user clicks **Run Testcase** in SilkTest, the following events occur:

1. For each script call, SilkTest contacts the SilkTest Agent and sends the information for the script call to be executed. This information includes the complete window declaration, the event name, and parameters.
2. The SilkTest Agent forwards that information to `FlexTechDomain`.
3. The `FlexTechDomain` uses `AutomationManager.resolveIDToSingleObject` with the Window Declaration information. The `AutomationManager` returns the resolved object based on the descriptive information, like `automationName`, `automationIndex`, `id`, and others.
4. Once the Flex control is resolved, `FlexTechDomain` calls `AutomationManager.replayAutomatableEvent()` to replay the event.
5. The `AutomationManager.replayAutomatableEvent()` method invokes the `IAutomationObject.replayAutomatableEvent()` method on the delegate class. The delegate uses the `IAutomationObjectHelper.replayMouseEvent()` method, or one of the other replay methods, such as `replayKeyboardEvent()`, to play back the event.
6. If there are verifications in your script, `FlexTechDomain` invokes `AutomationManager.getProperties()` to access the values that must be verified.

Testing the SilkTest Component Explorer Flex Sample Application

SilkTest provides a sample Adobe Flex test application called the Component Explorer. You must access the sample application at *http://demo.borland.com/flex/SilkTest2011/3.5/Flex3TestApp_withAutomation/Flex3TestApp.html*.

To test the Component Explorer, follow the steps described in the following topics:

- *Prerequisites for Testing Adobe Flex Applications*
- *Launching the Component Explorer*
- *Creating a new project*
- *Configuring Web Applications*
- *Recording a Sample Testcase for the Component Explorer*
- *Running a testcase*
- *Customizing Adobe Flex Scripts*

Alternatively, in the Help, click the **Contents** tab and then expand **Testing in Your Environment** > **Testing Adobe Flex Applications** > **Testing the Component Explorer Sample Application**. Follow the topics in the *Testing the Component Explorer Sample Application* section sequentially to test the Component Explorer sample application using SilkTest.

SilkTest provides several sample Flex applications. To access the samples, go to *http://demo.borland.com/flex/SilkTest2011/index.html* and choose the sample application you want to use.

Configuring Security Settings for Your Local Flash Player

Before you launch an Adobe Flex application, that runs as a local application, for the first time, you must configure security settings for your local Flash Player. You must modify the Adobe specific security settings to enable the local application access to the file system.

To configure the security settings for your local Flash player:

1. Open the **Flex Security Settings Page** by clicking **Flash Player Security Manager** on *http://demo.borland.com/flex/SilkTest2011/index.html*.
2. Click **Always allow**.
3. In the **Edit Locations** menu, click **Add Location**.
4. Click **Browse for folder** and navigate to the folder where your local application is installed.
5. Click **Confirm** and then close the browser.

Launching the Component Explorer

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

Before you launch the application for the first time, you must configure security settings for your local Flash Player.

To launch the Component Explorer in Internet Explorer, open *http://demo.borland.com/flex/SilkTest2011/3.5/Flex3TestApp_withAutomation/Flex3TestApp.html*.

The application launches in Internet Explorer.

Creating a New Project

You can create a new project and add the appropriate files to the project, or you can have SilkTest Classic automatically create a new project from an existing file.

Since each project is a unique testing environment, by default new projects do not contain any settings, such as extensions, class mappings, or Agent options. If you want to retain the settings from your current test set, save them as an options set by opening SilkTest Classic and clicking **Options** > **Save New Options Set**. You can add the options set to your project.

To create a new project:

1. In SilkTest Classic, click **File** > **New Project**, or click **Open Project** > **New Project** on the basic workflow bar.
2. On the **New Project** dialog box, click the type of project that you want to test.

   The type of project that you select determines the default Agent. For instance, if you specify that you want to create an Adobe Flex project, the Open Agent is automatically set as the default agent. SilkTest Classic uses the default agent when recording a test case, enabling extensions, or configuring an application.
3. Click **OK**.
4. On the **Create Project** dialog box, type the **Project Name** and **Description**.
5. Click **OK** to save your project in the default location, `<SilkTest Classic installation directory>\Project`.

   To save your project in a different location, click **Browse** and specify the folder in which you want to save your project.

   SilkTest Classic creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` files to this location and copies the extension .ini files, which are `appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini` to a `projectname\extend` subdirectory. SilkTest

Classic then creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources associated with this project.

6. Perform one of the following steps:
   - If your test uses the Open Agent, configure the application to set up the test environment.
   - If your test uses the Classic Agent, enable the appropriate extensions to test your application.

Configuring Web Applications

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a testcase. If you are testing a Flex application, use this configuration.

1. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
2. Double-click **Web Site Test Configuration**.
   The **New Web Site Configuration** page opens.
3. From the **Browser Type** list box, select **Internet Explorer**.

   You can use Firefox to replay tests but not to record them.
4. Perform one of the following steps:
   - Click **Use existing browser** to use a browser window that is already open to configure the test. For example, if the Web page that you want to test is already displayed in the browser window, you might want to use this option.
   - Click **Start new browser** to start a new browser instance to configure the test. Then, in the **Browse to URL** text box specify the Web page to open.
5. Click **Finish**.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
6. Navigate to the location in which you want to save the frame file.
7. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

   SilkTest automatically creates a base state for the application. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box. For instance, if you configure an application that uses an Internet Explorer or Firefox browser, SilkTest adds the `xBrowser.inc` file to the **Runtime Options** dialog box.

   SilkTest opens the Web page.
8. Record the test case whenever you are ready.

Recording a Sample Testcase for the Component Explorer

Use the following procedure to become familiar with the sample SilkTest Flex application, the Component Explorer.

To record a testcase for the Component Explorer:

1. Click **Record Testcase** on the Basic Workflow bar.
2. In the **Record Testcase** dialog box, type the name of your testcase in the **Testcase name** text box.

   Testcase names are case sensitive; they can have any length and consist of any combination of alphabetic characters, numerals, and underscore characters.

3. From the **Application State** list box, select **DefaultBaseState** to have the built-in recovery system restore the default BaseState before the testcase begins executing.
4. Click **Start Recording**.
   SilkTest closes the **Record Testcase** dialog box and displays the Flex sample application.
5. When the **Record Status** window opens, record the following scenario using the Flex sample application.

   It is essential that you perform these steps exactly as they are documented. Otherwise, your testcase script may not match the sample provided later in this document.
6. Click the ▶ arrow next to the **Visual Components** tree element to expand the list.
7. Click the ▶ arrow next to the **General Controls** tree element to expand the list.
8. Click the **SimpleAlert** tree element.
9. In the **Alert Control Example** section, click **Click Me** near the top of the window and then click **OK** in the **Hello World** message box.
10. Click the ▼ arrow next to the **General Controls** tree element to hide the list.
11. Click the ▼ arrow next to the **Visual Components** tree element to hide the list.
12. In the **Recording Status** window, click **Stop Recording**.
    SilkTest opens the **Record Testcase** dialog box, which contains the script that has been recorded for you.
13. Click **Paste to Editor**.
    The **Update Files** dialog box opens.
14. Choose **Paste testcase and update window declaration(s)** and then click **OK**.

Your testcase should include the following calls:

```
WebBrowser.BrowserWindow.Application.CompLibTree.Open("Visual Components")
WebBrowser.BrowserWindow.Application.CompLibTree.Open("Visual
Components>General Controls")
WebBrowser.BrowserWindow.Application.CompLibTree.Select("Visual
Components>General Controls>SimpleAlert")
WebBrowser.BrowserWindow.Application.Button1.Click()
WebBrowser.BrowserWindow.Application.Ok.Click()
WebBrowser.BrowserWindow.Application.CompLibTree.Close("Visual
Components>General Controls")
WebBrowser.BrowserWindow.Application.CompLibTree.Close("Visual Components")
```

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

Running a testcase

When you run a testcase, SilkTest interacts with the application by executing all the actions you specified in the testcase and testing whether all the features of the application performed as expected.

SilkTest always saves the suite, script, or testplan before running it if you made any changes to it since the last time you saved it. By default, SilkTest also saves all other open modified files whenever you run a script, suite, or testplan. To prevent this automatic saving of other open modified files, uncheck the **Save Files Before Running** check box in the **General Options** dialog.

1. Make sure that the testcase you want to run is in the active window.

2. Click **Run Testcase** on the **Basic Workflow** bar. If the workflow bar is not visible, choose **Workflows** > **Basic** to enable it.
3. SilkTest displays the **Run Testcase** dialog, which lists all the testcases contained in the current script.
4. Select a testcase and specify arguments, if necessary, in the **Arguments** field. Remember to separate multiple arguments with commas.
5. To wait one second after each interaction with the application under test is executed, check the **Animated Run Mode (Slow-Motion)** check box. Typically, you will only use this check box if you want to watch the testcase run. For instance, if you want to demonstrate a testcase to someone else, you might want to check this check box. Executions of the default base state and functions that include one of the following strings are not delayed:

   - Verify
   - Exists
   - Is
   - Get
   - Set
   - Print
   - ForceActiveXEnum
   - Wait
   - Sleep
6. To view results using the **Silk TrueLog Explorer**, check the **Enable TrueLog (for Classic Agent only)** check box. Click **TrueLog Options** to set the options you want to record.

   The **Silk TrueLog Explorer** works with the SilkTest Classic Agent only. Use the **Difference Viewer** to analyze results for testcases that use the SilkTest Open Agent.
7. Click **Run**. SilkTest runs the testcase and generates a results file.

   For the SilkTest Classic Agent, multiple tags are supported by Windows 2000 and Windows XP Agents. If you are running testcases using other Agents, you can run scripts that use declarations with multiple tags. To do this, check the **Disable Multiple Tag Feature** check box in the **Agent Options** dialog on the Compatibility tab. When you turn off multiple-tag support, 4Test discards all segments of a multiple tag except the first one.

Customizing Adobe Flex Scripts

You can manually customize your Flex scripts. You can insert verifications using the **Verification** wizard. Or, you can insert verifications manually using the `Verify` function on Flex object properties.

To customize Adobe Flex scripts:

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

   For example, the following code adds a verification call to your script:

```
Desktop.Find("//BrowserApplication").Find("//BrowserWindow")
.Find("//FlexApplication[@caption='explorer']").Find("//
FlexButton[@caption='OK']")
.VerifyProperties({...})
```

Each Flex object has a list of properties that you can verify. For a list of the properties available for verification, review the `Flex.inc` file. To access the file, navigate to the `<SilkTest directory>\extend\Flex` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\Flex\Flex.inc`.

Testing Flex Custom Controls

SilkTest supports testing Flex custom controls. By default, SilkTest provides record and playback support for the individual sub-controls of the custom control.

For testing custom controls, the following options exist:

Option	Description
Basic support	With basic support, you use dynamic invoke to interact with the custom control during replay. Use this low-effort approach when you want to access properties and methods of the custom control in the test application that SilkTest does not expose. The developer of the custom control can also add methods and properties to the custom control specifically for making the control easier to test. A SilkTest user can then call those methods or properties using the dynamic invoke feature. The advantages of basic support include: • Dynamic invoke requires no code changes in the test application. • Using dynamic invoke is sufficient for most testing needs. The disadvantages of basic support include: • No specific class name is included in the locator. For example, SilkTest records `//FlexBox` rather than `//FlexSpinner`. • Only limited recording support. • SilkTest cannot replay events. For more details about dynamic invoke, including an example, see *Dynamically Invoking Adobe Flex Methods*.
Advanced support	With advanced support, you create specific automation support for the custom control. This additional automation support provides recording support and more powerful play-back support. The advantages of advanced support include: • High-level recording and playback support, including the recording and replaying of events. • SilkTest treats the custom control exactly the same as any other built-in Flex control. • Seamless integration into SilkTest API. • SilkTest uses the specific class name in the locator. For example, SilkTest records `//FlexSpinner`. The disadvantages of advanced support include: • Implementation effort is required. The test application must be modified and the Open Agent must be extended.

Defining a Custom Control in the Test Application

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

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

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

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

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

To define the custom control:

1. In the test application, define the layout of the control.

   For example, for the spinner control type:

```
<?xml version="1.0" encoding="utf-8"?>
<customcontrols:SpinnerClass xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:controls="mx.controls.*" xmlns:customcontrols="customcontrols.*">
  <controls:Button id="downButton" label="Down" />
  <controls:TextInput id="text" enabled="false" />
  <controls:Button id="upButton" label="Up"/>
</customcontrols:SpinnerClass>
```

2. Define the implementation of the custom control.

   For example, for the spinner control type:

```
package
customcontrols
{
  import flash.events.MouseEvent;
  import mx.containers.HBox;
  import mx.controls.Button;
  import mx.controls.TextInput;
  import mx.core.UIComponent;
  import mx.events.FlexEvent;
  [Event(name="increment", type="customcontrols.SpinnerEvent")]
  [Event(name="decrement", type="customcontrols.SpinnerEvent")]

  public class SpinnerClass extends HBox
  {
    public var downButton : Button;
    public var upButton : Button;
    public var text : TextInput;
    public var ssss: SpinnerAutomationDelegate;
    private var _lowerBound : int = 0;
    private var _upperBound : int = 5;
    private var _value : int = 0;
    private var _stepSize : int = 1;

    public function SpinnerClass()
    {
      addEventListener(FlexEvent.CREATION_COMPLETE,
creationCompleteHandler);
    }

    private function creationCompleteHandler(event:FlexEvent) : void
    {
      downButton.addEventListener(MouseEvent.CLICK, downButtonClickHandler);
      upButton.addEventListener(MouseEvent.CLICK, upButtonClickHandler);
      updateText();
    }

    private function downButtonClickHandler(event : MouseEvent) : void
    {
      if(currentValue - stepSize >= lowerBound)
      {
        currentValue = currentValue - stepSize;
```

```
      }
      else
      {
        currentValue = upperBound - stepSize + currentValue - lowerBound +
1;
      }
      var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.DECREMENT);
      spinnerEvent.steps = _stepSize;
      dispatchEvent(spinnerEvent);
    }

    private function upButtonClickHandler(event : MouseEvent) : void
    {
      if(currentValue <= upperBound - stepSize)
      {
        currentValue = currentValue + stepSize;
      }
      else
      {
        currentValue = lowerBound + currentValue + stepSize - upperBound -
1;
      }
      var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.INCREMENT);
      spinnerEvent.steps = _stepSize;
      dispatchEvent(spinnerEvent);
    }

    private function updateText() : void
    {
      if(text != null)
      {
        text.text = _value.toString();
      }
    }

    public function get currentValue() : int
    {
      return _value;
    }

    public function set currentValue(v : int) : void
    {
      _value = v;
      if(v < lowerBound)
      {
        _value = lowerBound;
      }
      else if(v > upperBound)
      {
        _value = upperBound;
      }
      updateText();
    }

    public function get stepSize() : int
    {
      return _stepSize;
    }

    public function set stepSize(v : int) : void
    {
      _stepSize = v;
```

```
    }

    public function get lowerBound() : int
    {
      return _lowerBound;
    }

    public function set lowerBound(v : int) : void
    {
      _lowerBound = v;
      if(currentValue < lowerBound)
      {
        currentValue = lowerBound;
      }
    }

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

    public function set upperBound(v : int) : void
    {
      _upperBound = v;
      if(currentValue > upperBound)
      {
        currentValue = upperBound;
      }
    }
  }
}
```

3. Define the events that the control uses.

   For example, for the spinner control type:

```
package customcontrols
{
  import flash.events.Event;

  public class SpinnerEvent extends Event
  {
    public static const INCREMENT : String = "increment";
    public static const DECREMENT : String = "decrement";

    private var _steps : int;

    public function SpinnerEvent(eventName : String)
    {
      super(eventName);
    }

    public function set steps(value:int) : void
    {
      _steps = value;
    }

    public function get steps() : int
    {
      return _steps;
    }
  }
}
```

4. Proceed to *Implement Automation Support*.

Testing a Custom Control Using Dynamic Invoke

SilkTest provides record and playback support for custom controls using dynamic invoke to interact with the custom control during replay. Use this low-effort approach when you want to access properties and methods of the custom control in the test application that SilkTest does not expose. The developer of the custom control can also add methods and properties to the custom control specifically for making the control easier to test.

To test a custom control using dynamic invoke:

1. To retrieve a list of supported dynamic methods for a control, use the `GetDynamicMethodList` method.
2. Call dynamic methods on objects with the `DynamicInvoke` method.
3. Call multiple dynamic methods on objects with the `DynamicInvokeMethods` method.
4. To retrieve a list of supported dynamic properties for a control, use the `GetPropertyList` method.
5. Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty` method.

Example

This example tests a spinner custom control that includes two buttons and a text box, as shown in the following graphic.

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

The custom control offers a public `CurrentValue` property that can be set and retrieved. The value in this example is 3.

To set the spinner's value to 4, type the following:

```
WINDOW spinner = Desktop.Find("//
FlexBox[@className=customcontrols.Spinner]")
spinner.SetProperty("CurrentValue", 4)
```

Testing a Custom Control Using Automation Support

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

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

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

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

1. Open an existing SilkTest Flex project or create a new project.
2. Click **File** > **New**.

   The **New File** dialog box opens.
3. Choose **4Test include** and then click **OK**.

   A new include file opens.
4. Type the custom control class information in the INC file and then click **Save**.

For example, the INC file for the `FlexSpinner` class looks like the following:

```
winclass FlexSpinner : FlexBox
  tag "[FlexSpinner]"
  builtin void Increment(INTEGER steps)
  builtin void Decrement(INTEGER steps)
  property stepSize
    builtin INTEGER Get()
  property lowerBound
    builtin INTEGER Get()
  property currentValue
    builtin INTEGER Get()
    builtin Set(INTEGER value)
  property upperBound
    builtin INTEGER Get()
```

5. Click **Options** > **Runtime Options** and in the **Use Files** field navigate to the custom control INC file.
6. Record and replay tests for the custom control.

Implementing Automation Support for a Custom Control

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

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

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

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

1. Implement automation support, which is the automation delegate, in ActionScript for the custom control.

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

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

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

  [Mixin]
  public class SpinnerAutomationDelegate extends BoxAutomationImpl
  {
    public static function init(root:DisplayObject) : void
    {
      //register delegate for the automation
      Automation.registerDelegateClass(Spinner, SpinnerAutomationDelegate);
    }
```

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

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

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

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

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

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

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

    override public function replayAutomatableEvent(event:Event):Boolean
    {
      if(event is SpinnerEvent)
      {
        var spinnerEvent : SpinnerEvent = event as SpinnerEvent;
        if(event.type == SpinnerEvent.INCREMENT)
        {
          return replayClicks(spinner.upButton, spinnerEvent.steps);
        }
        else if
        {
          return replayClicks(spinner.downButton, spinnerEvent.steps);
        }
        else
        {
          return false;
        }
      }
      else
```

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

    // do not expose the child controls, which are the buttons and the
textfield, as individual controls
    override public function get numAutomationChildren():int
    {
      return 0;
    }
  }
}
```

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<TypeInformation>
  <ClassInfo Name="FlexSpinner" Extends="FlexBox">
    <Implementation Class="customcontrols.Spinner" />
    <Events>
      <Event Name="Decrement">
        <Implementation Class="customcontrols.SpinnerEvent"
          Type="decrement" />
        <Property Name="steps">
          <PropertyType Type="integer" />
        </Property>
      </Event>
    </Events>
    <Properties>
      <Property Name="lowerBound" accessType="read">
        <PropertyType Type="integer" />
      </Property>
      <Property Name="upperBound" accessType="read">
        <PropertyType Type="integer" />
      </Property>
      <!-- expose read and write access for the currentValue property -->
      <Property Name="currentValue" accessType="both">
        <PropertyType Type="integer" />
      </Property>
      <Property Name="stepSize" accessType="read">
        <PropertyType Type="integer" />
      </Property>
    </Properties>
  </ClassInfo>
</TypeInformation>
```

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

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

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

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

Now, you can test the custom control using SilkTest.

Flex Class Definition File

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

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

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

The XML file has the following basic structure:

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

Testing Web Applications

Using the xBrowser TechDomain

Overview of the xBrowser TechDomain

SilkTest provides support for testing Web applications using the Open Agent. Use the xBrowser technology domain to test Web applications that use:

- Internet Explorer
- Firefox
- Embedded browser controls

The xBrowser technology domain supports the testing of plain HTML pages as well as AJAX pages. AJAX pages require additional, sophisticated strategies for object recognition and synchronization.

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

Test Objects

SilkTest uses the following classes to model a Web application:

- `BrowserApplication` exposes the main window of a Web browser and provides methods for tabbing.
- `BrowserWindow` provides access to tabs and embedded browser controls and provides methods for navigating to different pages.
- `DomElements` exposes the DOM tree of a Web application (including frames) and provides access to all DOM attributes. Specialized classes are available for several DOM elements.

For a complete list of the record and replay controls available for xBrowser testing, see *xBrowser Class Reference*.

Object Recognition

The xBrowser technology domain supports dynamic object recognition.

When you record a testcase with the Open Agent, SilkTest creates locator keywords in an `INC` file to create scripts that use dynamic object recognition and window declarations.

Existing testcases that use dynamic object recognition without locator keywords in an `INC` file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object recognition without locator keywords in an INC file.

Testcases use locator strings to find and identify objects. A typical locator includes a locator name and at least one locator attribute, such as `"//LocatorName[@locatorAttribute='value']"`.

- Locator Names

  With other technology types, such as Java SWT, locator names are created using the class name of the test object. With xBrowser, the tag name of the DOM element can also be used as locator name. The following locators describe the same element.

  1. Using the tag name: `"//a[@href='http://www.microfocus.com']"`

  2. Using the class name: `"//DomLink[@href='http://www.microfocus.com']"`

  To optimize replay speed, use tag names rather than class names.
- Locator Attributes

  All DOM attributes can be used as locator string attributes. For example, the element `<button automationid='123'>Click Me</button>` can be identified using the locator `"//button[@automationid='123']"`.
- Recording Locators

  SilkTest uses a built-in locator generator when recording testcases and using the Locator Spy. You can configure the locator generator to improve the results for a specific application.

Page Synchronization

You can test standard HTML pages as well as AJAX pages with xBrowser. Configure the xBrowser synchronization settings to support the type of page that your environment requires. By default, AJAX synchronization is turned on, which enables the SilkTest Agent to wait until the Web page is in a valid state before the next action executes. As a result, no manual synchronization is necessary typically.

To use HTML page synchronization rather than AJAX synchronization, use the synchronization methods that SilkTest provides. Methods include:

- `WaitForObject`
- `WaitForProperty`
- `WaitForDisappearance`
- `WaitForChildDisappearance`

Regardless of the page synchronization method that you use, in tests where a Flash object retrieves data from a server and then performs calculations to render the data, you must manually add a synchronization

method to your testcase. Otherwise, SilkTest does not wait for the Flash object to complete its calculations. For example, you might add `Sleep(secs)`to your test.

Sample Applications

To access the SilkTest sample Web applications, go to:

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

SilkTest also provides a sample Java SWT test application that contains an embedded browser. You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java SWT** > **SWT Test Application <version>**. Select the sample application version that is right for your environment. Choose **Control/Standard Ctrl Sample** and then click the **Browser** tab.

Testing a Web Application using the xBrowser TechDomain

SilkTest provides built-in support for testing web applications with the xBrowser TechDomain. To test a Web application, follow these steps:

- Create a new project.
- Configure Web Applications.
- Record Testcases With the Open Agent
- Run a Testcase

SilkTest also provides sample web applications. To access the samples, go to:

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

For up-to-date information about supported versions, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes** to view the release notes.

xBrowser Default BaseState

By default, SilkTest uses the dynamic base state for xBrowser projects. When you configure the application, the base state is generated to the `frame.inc` file.

- The `wDynamicMainWindow` variable in the first line of the `frame.inc` file tells SilkTest to use the dynamic base state rather than the classic base state.
- The `WebBrowser` window declaration contains the necessary information to launch the browser and navigate to the Web application that you want to test.
- If you do not want to close all other tabs during base state execution, change `bCloseOtherTabs` to false.

Attributes for xBrowser 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.

The attributes that SilkTest supports for xBrowser applications include:

caption	Supports wildcards ? and *.
all DOM attributes	Supports wildcards ? and *.
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.

Attribute names are case sensitive.

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

For example:

```
<a>abc
    abc</a>
    //Uses the following locator:
//A[@innerText='abc abc']
```

Page Synchronization for xBrowser

Synchronization is performed before and after every method call. This means that the method call is not started and does not end until the synchronization criteria is met.

Any property access is not synchronized.

Synchronization Modes

SilkTest includes synchronization modes for HTML and AJAX.

Using the HTML mode ensures that all HTML documents are in an interactive state. With this mode, you can test simple Web pages. If more complex scenarios with Java script are used, it might be necessary to manually script synchronization functions, such as:

- `WaitForObject`
- `WaitForProperty`
- `WaitForDisappearance`
- `WaitForChildDisappearance`

The AJAX mode synchronization waits for the browser to be in a kind of idle state, which is especially useful for AJAX applications or pages that contain AJAX components. Using the AJAX mode eliminates the need to manually script synchronization functions (such as wait for objects to appear or disappear, wait for a specific property value, and so on), which eases the script creation process dramatically. This automatic synchronization is also the basis for a successful record and replay approach without manual script adoptions.

Troubleshooting

Because of the true asynchronous nature of AJAX, generally there is no real idle state of the browser. Therefore, in rare situations, SilkTest will not recognize an end of the invoked method call and throws a timeout error after the specified timeout period. In these situations, it is necessary to set the synchronization mode to HTML at least for the problematic call.

Regardless of the page synchronization method that you use, in tests where a Flash object retrieves data from a server and then performs calculations to render the data, you must manually add a synchronization method to your testcase. Otherwise, SilkTest does not wait for the Flash object to complete its calculations. For example, you might add `Sleep(secs)` to your test.

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. To add the URL to the exclusion filter, specify the URL in the **Synchronization exclude list** in the **Agent Options** dialog box.

Use a monitoring tool to determine if playback errors occur because of a synchronization issue. For instance, you can use FindBugs, *http://findbugs.sourceforge.net/*, to determine if an AJAX call is affecting playback. Then, add the problematic service to the **Synchronization exclude list**.

Note: If you exclude a URL, synchronization is turned off for each call that targets the URL that you specified. Any synchronization that is needed for that URL must be called manually. For example, you might need to manually add `WaitForObject` to a script. To avoid numerous manual calls, exclude URLs for a concrete target, rather than for a top-level URL, if possible.

Configuring Page Synchronization Settings

You can configure page synchronization settings for each individual test or you can set global options that apply to all tests in the Agent Options dialog box. Choose Options/Agent and click the **Synchronization** tab to configure these options.

To configure individual settings for tests, record the test and then insert an agent option to override the global replay value.

For example, you might set the Synchronization mode replay setting to HTML and then return the Synchronization mode to AJAX for the remaining portion of the test if necessary.

To configure individual settings within a test, call any of the following:

- `OPT_XBROWSER_SYNC_MODE`
- `OPT_XBROWSER_SYNC_EXCLUDE_URLS`
- `OPT_XBROWSER_SYNC_TIMEOUT`

Configuring the Locator Generator for xBrowser

The Open Agent includes a sophisticated locator generator mechanism that guarantees locators are unique at the time of recording and are easy to maintain. Depending on your application and the frameworks that you use, you might want to modify the default settings to achieve the best results.

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

To achieve optimal results, add a custom automation ID to the elements that you want to interact with in your test. In Web applications, you can add an attribute to the element that you want to interact with, such as `<div myAutomationId=″my unique element name″ />`. This approach can eliminate the maintenance associated with locator changes.

1. Click **Options** > **Recorder** and then click the **Custom Attributes** tab.
2. If you use custom automation IDs, from the **Select a TechDomain** list box, select **xBrowser** and then add the IDs to the list.

   The custom attributes list contains attributes that are suitable for locators. If custom attributes are available, the locator generator uses these attributes before any other attribute. The order of the list also represents the priority in which the attributes are used by the locator generator. If the attributes that you specify are not available for the objects that you select, SilkTest uses the default attributes for xBrowser.
3. Click the **Browser** tab.
4. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording.

   For example, use this list to specify attributes that change frequently, such as size, width, height, and style. You can include the wildcards ‘*’ and ‘?’ in the Locator attribute name blacklist.

   Separate attribute names with a comma.
5. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording.

   Some AJAX frameworks generate attribute values that change every time the page is reloaded. Use this list to ignore such values. You can also use wildcards in this list.

   Separate attribute values with a comma.
6. Click **OK**.

You can now record or manually create a test case.

Comparing API Replay and Native Replay for xBrowser

SilkTest supports API replay and native replay for Web applications. If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using API replay.

Advantages of native replay

- With native replay, the agent emulates user input by moving the mouse pointer over elements and pressing the corresponding elements. As a result, replay works with most applications without any modifications.
- Native replay supports plug-ins, such as Flash and Java applets, and applications that use AJAX, while high-level API recordings do not.

Advantages of API replay

- With API replay, the Web page is driven directly by DOM events, such as `onmouseover` or `onclick`.
- Scripts that use API replay do not require that the browser be in the foreground.
- Scripts that use API replay do not need to scroll an element into view before clicking it.
- Generally API scripts are more reliable since high-level user input is insensitive to pop-up windows and user interaction during replay.
- API replay is faster than native playback.

You can also use the Recording Options dialog box to configure the types of functions to record. Check the **OPT_XBROWSER_RECORD_LOWLEVEL** check box to use native user input instead of DOM functions.

Differences Between API and Native Replay Functions

The `DomElement` class provides different functions for API replay and native replay.

The following table describes which functions use API replay and which use native replay.

	API Replay	Native Replay

Mouse Actions	`DomClick`
	`DomDoubleClick`
	`DomMouseMove`
		`Click`
		`DoubleClick`
		`MouseMove`
		`MousePress`
		`MouseRelease`

Keyboard Actions	not available	`TypeKeys`

Specialized Functions	`Select`
	`SetText`
	etc.
		not available

Setting Recording Options for xBrowser

There are several options that can be used to optimize the recording of Web applications.

1. Click **Options** > **Recorder**.
2. Check the **Record mouse move actions** box if you are testing a web page that uses mouse move events.

SilkTest Classic will only record mouse move events that cause changes to the hovered element or its parent in order to keep scripts short.

3. You can change the **mouse move delay** if required.

   Mouse move actions will only be recorded if the mouse stands still for this time. A shorter delay will result in more unexpected mouse move actions.
4. Click the **Browser** tab.
5. In the **Locator attribute name exclude list** grid, type the attribute names to ignore while recording.

   For example, if you do not want to record attributes named **height**, add the **height** attribute name to the grid. Separate attribute names with a comma.
6. In the **Locator attribute value exclude list** grid, type the attribute values to ignore while recording.

   For example, if you do not want to record attributes assigned the value of **x-auto**, add **x-auto** to the grid. Separate attribute values with a comma.
7. To record native user input instead of DOM functions, check the **OPT_XBROWSER_RECORD_LOWLEVEL** check box.

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

   If your application uses a plug-in or AJAX, use native user input. If your application does not use a plug-in or AJAX, we recommend using high-level DOM functions, which do not require the browser to be focused or active during playback. As a result, tests that use DOM functions are faster and more reliable.
8. Click the **Custom Attributes** tab.
9. Select **xBrowser** in the **Select a tech domain** list box and add the DOM attributes that you want to use for locators to the text box.

   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. 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, SilkTest Classic uses the default attributes for xBrowser.
10. Click **OK**.

    You can now record or manually create a test that uses ignores browser attributes and uses the type of page input that you specified.

Browser Configuration Settings for xBrowser

Several browser settings help to sustain stable test executions. Although SilkTest works without changing any settings, there are several reasons that you might want to change the browser settings.

Increase replay speed	Use `about:blank` as home page instead of a slowly loading web page
Avoid unexpected behavior of the browser	• Disable pop up windows and warning dialogs • Disable auto complete features • Disable password wizards
Prevent malfunction of the browser	Disable unnecessary third-party plugins

The following sections describe where these settings are located in Internet Explorer and Firefox.

Internet Explorer

The browser settings are located at **Tools** > **Internet Options**. The following table lists options that you might want to adjust.

Tab	Option	Configuration	Comments
General	Home page	Set to "about:blank"	Minimize start up time of new tabs.
General	Tabs	• Disable warning when closing multiple tabs • Enable to switch to new tabs when they are created	• Avoid unexpected dialogs. • Links that open new tabs might not replay correctly otherwise.
Privacy	Pop-up blocker	Disable pop up blocker	Make sure your Web site can open new windows.
Content	AutoComplete	Turn off completely	• Avoid unexpected dialogs. • Avoid unexpected behavior when typing keys.
Programs	Manage add-ons	Only enable add-ons that are absolutely required	• Third-party add-ons might contain bugs. • Possibly not compatible to SilkTest.
Advanced	Settings	• Disable "Automatically check for Internet Explorer updates" • Enable "Disable script debugging (Internet Explorer)" • Enable "Disable script debugging (Other)" • Disable "Enable automatic crash recovery" • Disable "Display notification about every script error" • Disable all "Warn ... " settings	Avoid unexpected dialogs.

Firefox

In Firefox, you can edit all settings by navigating a tab to `about:config`. The following table lists options that you might want to adjust. If any of the options do not exist, you can create them by right-clicking the table and choosing **New**.

Option	Value	Comments
app.update.auto	false	Avoid unexpected behavior (disable auto update).
app.update.enabled	false	Avoid unexpected behavior (disable updates in general).
app.update.mode	0	Avoid unexpected dialogs (do not prompt for new updates).
app.update.silent	true	Avoid unexpected dialogs (do not prompt for new updates).
browser.sessionstore.resume_from_crash	false	Avoid unexpected dialogs (warning after a browser crash).
browser.sessionstore.max_tabs_undo	0	Enhance performance. Controls how many closed tabs are kept track of through the Session Restore service.
browser.sessionstore.max_windows_undo	0	Enhance performance. Controls how many closed windows are kept track of through the Session Restore service.

Option	Value	Comments
browser.sessionstore.resume_session_once	false	Avoid unexpected dialogs. Controls whether the last saved session is restored once the next time the browser starts.
browser.shell.checkDefaultBrowser	false	Avoid unexpected dialogs (checks if Firefox is the default browser).
browser.startup.homepage	"about:blank"	Minimize start up time of new tabs.
browser.startup.page	0	Minimize browser startup time (no start page in initial tab).
browser.tabs.warnOnClose	false	Avoid unexpected dialogs (warning when closing multiple tabs).
browser.tabs.warnOnOpen	false	Avoid unexpected dialogs (warning when opening multiple tabs).
dom.max_chrome_script_run_time	180	Avoid unexpected dialogs (warning when XUL code takes too long to execute, timeout in seconds).
dom.max_script_run_time	600	Avoid unexpected dialogs (warning when script code takes too long to execute, timeout in seconds).
extensions.update.enabled	false	Avoid unexpected dialogs. Disables automatic extension update.

Changing the Browser Type During Replay

You can replay testcases in either Internet Explorer or Firefox when testing Web applications.

1. Configure the web application that you want to test using Internet Explorer. Tests can only be replayed with Firefox at this time.
2. Record the testcase and replay it to ensure it works as expected.
3. Click **Configure Application** on the **Basic Workflow** bar.
   The **New Test Frame** dialog box opens.
4. Double-click **Web Site Test Configuration**.
   The **New Web Site Configuration** page opens.
5. From the **Browser Type** list, select `Mozilla Firefox`.
6. Perform one of the following steps:

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

7. Click **Finish**.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame1.inc` by default.
8. Navigate to the location in which you want to save the frame file.
9. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.
10. Copy the window declaration for Firefox into the original `frame.inc` file. The `frame.inc` must not contain two window declarations of the same name, so rename the Internet Explorer declaration to InternetExplorer and the Firefox declaration to Firefox.

11. To ensure that each of the window declarations works with the appropriate browser type, add the `browsertype` property to the locator, for example, `// BrowserApplication[@browsertype='Firefox']`.
12. Determine which browser you want to use and change the const `wDynamicMainWindow = <browsertype>` to use that browser and then replay the test.

    For example, you might type const `wDynamicMainWindow = InternetExplorer`

xBrowser Frequently Asked Questions

This topic includes a collection of questions that you might encounter when testing your web application with SilkTest.

How can I verify the font type that is used for the text of an element?

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

In Internet Explorer, use `wDomElement.GetProperty("currentStyle:fontName")`. In Firefox, use `wDomElement.GetProperty("currentStyle:font-name")`.

What is the difference between textContents, innerText, and innerHtml?

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

Consider the following html code:

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

The following table details the different properties that return.

Code	Returned Value
`Desktop.Find("//div[@id='mylinks']").GetProperty("textContents")`	This is my link collection:
`Desktop.Find("//div[@id='mylinks']").GetProperty("innerText")`	This is my link collection:Bye bye Borland Welcome to Micro Focus

`Desktop.Find("//div[@id='mylinks']").GetProperty("innerHtml")`	`This is my <b>link collection</b>:`
	`<ul>`
	`  <li>`
	`  <a href="www.borland.com">Bye bye`
	`<b>Borland</b> </a>`
	`  </li>`

Code	Returned Value
	 Welcome to Micro Focus

I configured innerText as a custom class attribute, but it is not used in locators

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

What should I take care of when creating cross browser scripts?

1. Different attribute values (for example, colors in Internet Explorer are returned as `#FF0000` and in Firefox as `rgb(255,0,0)`)
2. Different attribute names (for example, the font name attribute in Internet Explorer is called `fontName` and in Firefox `font-name`)
3. Some frameworks may render different DOM trees

How can I distinguish Firefox from Internet Explorer in my scripts

1. The `BrowserApplication` class provides a property `browsertype` that returns either `Firefox` or `Internet Explorer`. You can add this to a locator in order to define which browser it matches.
2. The `BrowserWindow` provides a method `GetUserAgent` that returns the user agent string of the current window.

Which locators are best suited for stable cross browser testing?

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

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

Logging output of my application contains wrong timestamps

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

My test script hangs after navigating to a new page

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

I recorded an incorrect locator

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

The rectangles around elements in Internet Explorer are misplaced

1. Make sure the zoom factor is set to 100%. Otherwise, the rectangles are not placed correctly.

2. Ensure that there is no notification bar displayed above the browser window. SilkTest cannot handle notification bars.

Link.Select does not set the focus for a newly opened window in Internet Explorer

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

DomClick(x, y) is not working like Click(x, y)

If your application uses the `onclick` event and requires coordinates, the `DomClick` method does not work. Try to use `Click` instead. For more information, see *Comparing API Replay and Native Replay*.

FileInputField.DomClick() will not open the dialog

Try to use `Click` instead. For more information, see *Comparing API Replay and Native Replay*.

The mouse moves setting is turned on but all mouse moves are not recorded. Why not?

In order to not pollute the script with a lot of useless `MouseMove` actions:

1. SilkTest will only record a MouseMove action if the mouse stands still for a specific time.
2. SilkTest only records MouseMove actions if it observes activity going on after an element was hovered over. In some situations, you might need to add some manual actions to your script.

I need some functionality that is not exposed by the xBrowser API. What can I do?

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

Why are the class and the style attributes not used in the locator?

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

Testing the SilkTest Insurance Company Sample Web Application

Testing the SilkTest Insurance Company Sample Web Application

SilkTest provides a sample insurance company web application, *http://demo.borland.com/InsuranceWebExtJS/*.

To complete a tutorial for how to test the insurance company web application using SilkTest, complete each of the following steps. Or, in the online help, click the **Contents** tab and then expand **Testing in Your Environment** > **Testing Web Applications** > **Using the xBrowser Tech Domain** > **Testing the SilkTest Insurance Company Sample Web Application**. Follow the topics sequentially in the **Testing the SilkTest Insurance Company Sample Web Application** book to test the sample web application using SilkTest.

To test the sample web application, follow these steps:

- Creating a New Project for the Insurance Company Web Application.
- Configuring the Insurance Company Web Application.
- Recording a Testcase for the Insurance Company Web Site.
- Replaying the Testcase for the Insurance Company Web Site.
- Modifying the Insurance Company Testcase to Replay Tests in Firefox.

If you install the SilkTest samples a complete sample project, which includes sample scripts, is available. Open the project by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Scripts** > **4Test** > **xBrowser** and opening the project file.

Creating a New Project for the Insurance Company Web Application

The type of project that you select determines the default SilkTest Agent. For Web application projects, the Open Agent is automatically set as the default agent. SilkTest uses the default agent when configuring an application and recording a testcase.

1. Click **File** > **New Project**, or click **Open Project** > **New Project** on the **Basic workflow** bar.

   The **New Project** dialog opens.
2. Under **Rich Internet Applications**, click **Web**.
3. Click **OK**.

   The **Create Project** dialog opens.
4. Type a project name and a description in the appropriate text boxes.
5. Click **OK** to save your project in the default location, `<SilkTest installation directory>\Project`.

   To save your project in the default location, click **Browse** and specify the folder in which you want to save your project.

   SilkTest creates a `projectname` folder within this directory, saves the `projectname.vtp` and `projectname.ini` to this location and copies the extension `.ini` files (`appexpex.ini`, `axext.ini`, `domex.ini`, and `javaex.ini`) to a `projectname\extend` subdirectory.

   SilkTest creates your project and displays nodes on the **Files** and **Global** tabs for the files and resources associated with this project.

Configuring the Insurance Company Web Application

When you configure an application, SilkTest automatically creates a base state for the application. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution.

1. Click **Configure Application** on the **Basic Workflow** bar.
   The New Test Frame dialog box opens.
2. Double-click **Web Site Test Configuration**.
   The **New Web Site Configuration** page opens.
3. From the **Browser Type** list, select **Internet Explorer**.

   You can use Firefox to replay tests but not to record them.
4. Perform one of the following steps:
   a) **Use existing browser**. Click this option button to use a browser window that is already open to configure the test. For example, if the web page that you want to test is already displayed in the browser window, you might want to use this option.
   b) **Start new browser**. Click this option button to start a new browser instance to configure the test. Then, in the **Browse to URL** text box specify the web page to open.

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

   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
6. Navigate to the location in which you want to save the frame file.
7. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

   SilkTest automatically creates a base state for the application. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files**

location in the **Runtime Options** dialog. For instance, if you configure an application that uses an Internet Explorer or Firefox browser, SilkTest adds the `xBrowser.inc` file to the **Runtime Options** dialog.

SilkTest opens the web page. Record the testcase whenever you are ready.

Recording a Testcase for the Insurance Company Web Site

1. Click **Record Testcase** on the **Basic Workflow** bar.
   The **Record Testcase** dialog box opens.
2. Type the name of your testcase in the Testcase name text box.

   For example, type `ZipTest`.

   Testcase names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
3. From the **Application State** list box, select **DefaultBaseState** to have the built-in recovery system restore the default base state before the testcase begins executing. If you choose **DefaultBaseState** as the application state, the testcase is recorded in the script file as: `testcase testcase_name ()`.
4. If you do not want SilkTest to display the status window during playback when driving the application to the specified base state, uncheck the **Show AppState status window** check box.

   Typically, you check this check box. However, in some circumstances it is necessary to hide the status window. For instance, the status bar might obscure a critical control in the application you are testing.
5. Click **Start Recording**. SilkTest:
6. In the insurance company web site, perform the following steps:
7. To review what you have recorded, click Stop Recording in the Recording window.

   SilkTest displays the **Record Testcase** dialog box, which contains the code that has been recorded for you.
8. Click **Paste to Editor**.
9. Choose **File** > **Save**.

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

Replaying the Testcase for the Insurance Company Web Site

Replay a test to ensure that it works as expected.

1. Make sure that the testcase you want to run is in the active window.
2. Click **Run Testcase** on the **Basic Workflow** bar.

   SilkTest displays the **Run Testcase** dialog, which lists all the testcases contained in the current script.
3. Select the ZipTest testcase.
4. To wait one second after each interaction with the application under test is executed, check the **Animated Run Mode (Slow-Motion)** check box. Typically, you will only use this check box if you want to watch the testcase run. For instance, if you want to demonstrate a testcase to someone else, you might want to check this check box. Executions of the default base state and functions that include one of the following strings are not delayed:
5. Click **Run**.

   SilkTest runs the testcase and generates a results file. The results file describes whether the test passed or failed, and provides summary information.

Modifying the Insurance Company Testcase to Replay Tests in Firefox

The original base state uses Internet Explorer as the browser. Create a base state for Firefox and add it to the existing base state file to replay tests on both Internet Explorer and Firefox.

1. Click **Configure Application** on the **Basic Workflow** bar.

The **New Test Frame** dialog box opens.

2. Double-click **Web Site Test Configuration**.

   The **New Web Site Configuration** page opens.

3. From the **Browser Type** list, select **Mozilla Firefox**.

   You can use Firefox 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 window that is already open to configure the test. For example, if the Web page that you want to test is already displayed in the browser window, you might want to use this option.
   - **Start new browser** - Click this option button to start a new browser instance to 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**.

   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame1.inc` by default.

6. Navigate to the location in which you want to save the frame file.
7. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Click **Save**.
8. Copy the window declaration for Firefox into the original `frame1.inc` file. `frame1.inc` must not contain two window declarations of the same name, so rename the Internet Explorer declaration to `InternetExplorer` and the Firefox declaration to `Firefox`.
9. To ensure that each of the window declarations works with the appropriate browser type, add the `browsertype` property to the locator, for example, `"//BrowserApplication[@browsertype='Firefox']"`.
10. Determine which browser you want to use and change the const wDynamicMainWindow = <browsertype> to use that browser and then replay the test.

    For example, you might type const `wDynamicMainWindow = InternetExplorer`

The following code shows a frame file that includes window declarations for Internet Explorer and Firefox. You can copy the following code by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Scripts/4Test Samples** and opening the `frame.inc` file.

frame.inc

```
[ ] // This frame.inc contains window declarations for Firefox and Internet
[ ] // Explorer.
[ ] // The wDynamicMainWindow variable indicates that the dynamic base state
[ ] // will be used. The window declaration that is assigned here will be
[ ] // used for launching the application and waiting for the main window.
[ ] // This is where you can decide whether to run your tests on Internet
[ ] // Explorer or on Firefox.
[ ] const wDynamicMainWindow = InternetExplorer
[ ]
[ ] // Window declaration for Internet Explorer
[-] window BrowserApplication InternetExplorer

[ ] // Use the browsertype property in order to ensure an Internet Explorer
[ ] // is running
[ ] locator "//BrowserApplication[@browsertype='Internet Explorer']"

[ ] // The working directory of the application when it is invoked
[ ] const sDir = "C:\Program Files\Internet Explorer"

[ ] // The command line used to invoke the application
```

```
[ ] const sCmdLine = "C:\Program Files\Internet Explorer\IEXPLORE.EXE"

[ ] // The start URL
[ ] const sUrl = "http://demo.borland.com/InsuranceWebExtJS"
[ ]
[ ] // Window declaration for Firefox
[-] window BrowserApplication Firefox

[ ] // Use the browsertype property in order to ensure a Firefox
[ ] // is running
[ ] locator "//BrowserApplication[@browsertype='Firefox']"

[ ] // The working directory of the application when it is invoked
[ ] const sDir = "C:\Program Files\Mozilla Firefox"

[ ] // The command line used to invoke the application
[ ] const sCmdLine = "C:\Program Files\Mozilla Firefox\firefox.exe"

[ ] // The start URL
[ ] const sUrl = "http://demo.borland.com/InsuranceWebExtJS"
```

xBrowser Classes

Using the Classic Agent

Web Application Support

SilkTest provides support for applications that require web testing capabilities, including web applications and browsers, using the SilkTest Open Agent by default. For example, web application support enables you to test an application that runs in Internet Explorer.

To create a web application that uses the Classic Agent, create a Generic Classic Agent project.

Projects that use the SilkTest Classic Agent support hierarchical object recognition only. As a best practice, we recommend using the Rich Internet Application web project rather than the Generic Classic Agent project type because the web application uses the SilkTest Open Agent and supports dynamic 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 testcase if necessary. Use the method best suited to meet your test requirements.

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

Supported Controls

For a complete list of the record and replay controls available for Web application testing, view the `browser.inc` and `explorer.inc` files. By default, these files are located in `C:\Program Files\Silk\SilkTest\extend\`. The `browser.inc` file contains the objects that are shared by all web browsers (such as the **Back** button on the toolbar). Objects that are unique to each browser are included in a separate file. Internet Explorer objects are contained in `explorer.inc`.

Overview of setup steps

Before testing a web application, take the following steps to set up SilkTest for this type of testing:

- Enable support for browsers and disable all non-Web extensions.
- Specify your default browser.
- Make sure your browser is configured properly.
- Set the proper Agent options, if necessary.

Enabling Extensions for Embedded Browser Applications

To test an embedded browser application, enable the Web browser as the primary extension for the application in both the **Extension Enabler** and in the **Extensions** dialog boxes. For instance, if you are testing an application with DOM controls that are embedded within a .NET application, follow these instructions to enable extensions.

1. Click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Extension Enabler** to start the **Extension Enabler**.
2. On the **Extension Enabler** dialog box, click **New**.
3. Click  to navigate to the location of the application executable.
4. Select the executable file and then click **Open**.
5. Click **OK**.
6. From the **Primary Extension** list box, select the DOM extension for the application that you added.
7. Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate.

   For example, to test a .NET application with embedded Web controls, select a browser in the **Primary Extension** list and check the **.NET** check box for the application within the grid.
8. Click **OK**.
9. Start SilkTest Classic and then choose **Options** > **Extensions**.
10. On the **Extensions** dialog box, click **New**.
11. Click  to navigate to the location of the application executable.
12. Select the executable file and then click **Open**.
13. Click **OK**.
14. From the **Primary Extension** list box, select the DOM extension for the application that you added.
15. Enable other extensions, such as Java, ActiveX, Accessibility, and .NET, as appropriate.

    For example, to test a .NET application with embedded Web controls, select a browser in the **Primary Extension** list and check the **.NET** check box for the application within the grid.
16. Click **OK**.
17. Re-start SilkTest Classic.

The IE DOM extension may not detect changes to a web page that occur when JavaScript replaces a set of elements with another set of elements without changing the total number of elements. To force the DOM extension to detect changes in this situation, call the `FlushCache()` method on the top-level browserchild for the embedded browser.

Note: This problem may occur more often for embedded browsers than for browser pages, because SilkTest is not notified of as many browser events for embedded browsers.

Also call `FlushCache()` if you get a `Coordinate out of bounds` exception when calling a method, for example `Click()`, on an object that previously had been scrolled into view.

Note: The `BrowserPage` window identifier is not valid when using embedded browsers because the default browser type is '(none)' (NULL).

Troubleshooting Test Failure for Web Applications that Use the Classic Agent

The test of your browser application may have failed for the one of the reasons described below. If the following suggestions do not address the problem you are having, you can enable your extension manually, see *Enabling Extensions*.

The application may not have been ready to test

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. Click **OK** on the **Extension Settings** dialog box, and then close and restart your application.
4. Make sure the application has finished loading, and then click **Test**.

The page you selected was empty or did not contain HTML elements

In this case, you may receive the following message: `Could not recognize any HTML classes in your browser application`. Your configuration may be correct, however, testing of blank pages or pages that do not contain HTML elements is not supported in the automated configuration test. You can manually verify that your extensions are set properly, open your application, and then record window declarations. If you can record against HTML classes, the extension is configured correctly and you are ready to *Set up the Recovery System* using the **Basic workflow** bar.

You cannot enable Firefox while Netscape is running, and vice versa

You may not enable Firefox 1.5 through the SilkTest Basic Workflow or otherwise work with Firefox if Netscape 7.0 is also running. Likewise, you may not enable Netscape 7.0 through SilkTest Basic Workflow or work with Netscape if Firefox 1.5 is running.

Note: Close one of the browsers and continue.

Recording the test frame for a web application

When you record a test frame for a Web application, the results differ from those for a non-web application.

1. Start your browser and go to the initial page of your web application.
2. Click **File** > **New** from the menu bar.
3. Click **Test Frame** and then click **OK**.
   The **New Test Frame** dialog box displays. It lists all open applications that are not minimized, including your web application, which is identified by the currently loaded page's title.
4. Select your web application. The dialog displays the following fields:

File name	Name of the frame file you are creating. You can change the name and path to anything you want, but make sure you retain the `.inc` extension.
Application	The title of the currently loaded page.
URL	The URL of the currently loaded page.
4Test identifier	The identifier that you will use in all your testcases to qualify your application's home page. Make it something meaningful (and short) for your application.

5. Edit the file name and 4Test identifier as appropriate.
6. Click **OK**.

Recording window declarations for a web application

Window declarations can be saved in a single file or in multiple files. If you save them in multiple files, make them available to scripts using the 4Test `use` statement.

1. Click the test frame to make it active.
2. Click **Record** > **Window Declarations**. SilkTest displays the **Record Window Declarations** dialog.
3. Load a page in your application in the browser.
4. Place your mouse pointer over the page. The value in the **Class** field should be `BrowserChild`, since that is the class for a page in a Web application.

5. Move your mouse pointer around the page. Notice that the values in the dialog update to reflect the object the mouse is over. Notice all the objects that SilkTest sees. Press `Ctrl+Alt`. The contents freeze in the **Record Window Declarations** dialog.
6. Click **Paste to Editor**. SilkTest pastes the new declarations to your frame file.
7. To declare another page of the application, go to that page. Then, in the **Record Window Declarations** dialog, click **Resume Tracking**.
8. Click **Close**.

You can now use multitags when recording window declarations for Java applications. However, additional considerations must be made for top-level windows.

Modifying the identifiers in the declaration	After you have declared each of your application's pages, you will probably want to modify the identifiers to be more meaningful.
Test frame for the GMO application	We provide a complete test frame for the sample GMO application, the `gmow.inc` file. You can download a web-based version of the GMO application at *http://demo.borland.com/gmopost/*.

Recording Dynamic HTML (DHTML) Popup Menus

You must enable extensions for the application that contains the JavaScript and use the Classic Agent to record dynamic popup menus.

If you want your action-based recordings to contain references to window identifiers instead of dynamic instantiations, first record the window declarations for the pages with DHTML popup menus. There are various techniques used to build DHTML popup menus and their menu hierarchies. The techniques you use affect what SilkTest sees when recording window declarations. You may find that once a page is completely loaded in the browser, all of the menus and submenus are recognized immediately by SilkTest. Other times, in order for the menus and submenus to be completely seen in the **Record Window Declarations** dialog box, you may need to expose some or all of the menus and submenus by moving the mouse over the menu items.

Typically when you record actions, the recorder ignores mouse movement events, which are set in the **Ignore Mouse Move Events** text box of the **Recorder Options** dialog box. However, the recorder generates `MoveMouse()` method calls as you expose popup menus. Those calls are necessary to ensure that when you play back the script, SilkTest exposes the menus as it navigates through them. The `MoveMouse()` calls contain coordinates because the hot spot of the item used to expose the menu may not be the entire rectangle for that item. Therefore, SilkTest cannot assume that moving the mouse to the default spot, which is the upper-left corner of the rectangle for the item, will actually expose the menu.

Streamlining HTML frame declarations

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

As you navigate within a web site that uses frames, the GUI objects in individual frames may change independently of other frames in the same window. When you capture declarations for the new GUI objects inside a frame, SilkTest redeclares the frame and the frame's own parent window. If all pages in the frame have the same caption, you will want to do the following:

1. Record a new test frame for the Web page.
   SilkTest captures all the active HTML frames as displayed on the browser.
2. To declare other HTML frames, make the page appear in the browser. This is usually done by clicking a link in an "index" type HTML frame. For example, a static HTML frame region may contain a menu bar or image map to navigate the Web site.
3. Open the newly recorded declaration, and locate the declaration for the new HTML frame. Copy this `BrowserChild` object and paste it into bottom of the declaration. This new `BrowserChild` is a sibling (at the same level) to the initial `BrowserChild` declarations recorded. Re-name this `BrowserChild` as desired for easier recognition.

4. Remove the recorded window declaration (remember you just copied the declaration for the new Html frame into the "`main/root`" `BrowserChild` declaration. This declaration and all its children should not be deleted.)

Test Frames

Tags and Identifiers

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Each object in a declarations file, such as a test frame file, has a class, a tag, and an identifier. The home page has the class `BrowserChild`. Its default identifier is the name in the 4Test identifier field you specified when you created the test frame. The tag is generated by SilkTest. It is the way that SilkTest identifies the page at runtime.

Overview of test frames

A test frame is an include file (`.inc`) that serves as a central global, repository of information about the application under test. It contains all the data structures that support your testcases and test scripts. Though you do not have to create a test frame, by declaring all the objects in your application, you will find it much easier to understand, modify, and interpret the results of your tests.

When you create a test frame, SilkTest automatically adds the frame file to the **Use files** field of the **Runtime Options** dialog. This allows SilkTest to use the information in the declarations and recognize the objects in your application when you record and run testcases.

When you enable extensions, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog. For extensions that use the Open Agent, SilkTest names the include file `<technology_type>.inc`. For instance, if you enable extensions for an Adobe Flex application, a file named `flex.inc` is added. If you enable extensions for an Internet Explorer browser, SilkTest adds the `explorer.inc` file to the **Runtime Options** dialog.

A constant called `wStartup` is created when you record the test frame. By assigning the identifier of the login window to `wStartup` and by recording a new `invoke` method, your tests can start the application, enter any required information into the login window, then dismiss the login window. For more information, see Handling login windows in non-Web applications.

See Marking 4Test code as GUI-specific to learn about the ways you modify the test frame when porting your testcases to other GUIs.

The test frame file for a Web application

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The test frame file includes the following:

- A constant named `wMainWindow`
- A window of class `BrowserChild`

wMainWindow

This constant points to the home page of your application, that is, the page that was loaded when you created the test frame. The recovery system uses `wMainWindow` to restore the browser to that page when a test fails. Just as a non-web application typically has a state where you want the tests to start (the base state), web applications also have a base state. Typically, it is the first page in the application. See *Web applications and the recovery system* for more information.

BrowserChild

The window has the same identifier as the value of `wMainWindow`. This window loads in order to restore the base state. The window declaration contains:

- The constant `sLocation`, which is the URL for the page. The recovery system uses this constant to load the page when necessary.

- Two commented constants, `sUserName` and `sPassword` which specify the user name and password to access the application. See *Specifying username and password*.
- Two commented constants, `BrowserSize` and `bDefaultFont`, which specify the size of the browser window and the default font to use for displaying text. See *Specifying browser size and fonts*.
- All the objects in the page, such as `HtmlHeadings`, `HtmlText`, `HtmlLinks`, `HtmlText`, `HtmlPushButtons`, and so on.

Specifying browser size and fonts

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

By default, two other built-in constants are also enclosed in comment tags in a generated test frame: `BrowserSize` and `bDefaultFont`. The recovery system uses these two constants to set the browser's state before and after SilkTest runs each test is run. They are useful in establishing a consistent environment for your testing.

BrowserSize

Specifies the width and height of the browser window, in pixels. The data type is `POINT`, which is a record of two integers. The first integer is the width in pixels. The second integer is the height in pixels. The default value ({600, 400}) is an appropriate size for a screen with VGA resolution (640 by 480).

If you are using a higher resolution, you would want a larger size for the browser window (the larger the better, in order to minimize scrolling in the testcases). For example, if you are using a resolution of 1280 by 1024, a good `BrowserSize` might be {650, 900}, which would take up about half the width and most of the height of the desktop.

bDefaultFont

If this constant is set to TRUE, the recovery system will restores the fonts to the shipped defaults for the browser, as described in `SetDefaultFont`.

Using these constants

To have the recovery system set the size and fonts, uncomment the constants in the test frame and specify appropriate values for `BrowserSize`.

Having the recovery system set the browser size and fonts ensures maximum portability of your window declarations in different testing sessions and between browsers. We strongly recommend that you uncomment these constants and use the recovery system for your Web testing.

Modifying the identifiers

Identifiers are arbitrary strings. You use identifiers to identify objects in your scripts. Tags, on the other hand, are not arbitrary and should not be changed except in well-specified ways.

To make your tests easier to understand and maintain, you can change your objects' identifiers to correspond to their meaning in your application. Then when SilkTest records tests, it will use the identifiers that you specify.

User Options

Setting DOM extension options

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

There are three different ways to set options for the DOM extension.

- In the **DOM Extensions** dialog.
- By using `BrowserPage.SetUserOption()` in a script.
- By editing the values in the Options section of the `domex.ini` file.

Depending on where you set the option, the option can be set globally, or turned on and off at various points of your testing.

Setting an option in `domex.ini` or in the **DOM Extensions** dialog sets it globally. However, if you want to set the option only at certain points in your script, use `BrowserPage.SetUserOption()` as described in `SetUserOption()`.

Regardless of how an option is set, you can read its value by using `GetUserOption()`.

Options you can set in `domex.ini`

To set these options, you enter them on a line in the [Options] section of `domex.ini`, for example:

```
ScrollListItemIntoView=FALSE
```

You can set the following options in `domex.ini`. You cannot set them with `SetUserOption()`.

- `DOMWaitForBrowser` – default is 10000 milliseconds (or 10 seconds). The value you set is expressed in milliseconds. This option specifies how long SilkTest will wait for the browser to complete an action. If the browser fails to respond within the given time, SilkTest will try to force a ready-state. If this fails too, an error will occur.
- `IgnoreDivTags` – default is FALSE. If HTML controls nested between the <DIV> and <\DIV> tags are not recognized, set this option to TRUE to ignore the <DIV> and <\DIV> tags.
- `ReturnListContentsPropertyAsString` – default is FALSE meaning that normally the `$Contents` property for `HtmlList` objects returns a LIST OF STRING. Set this option to TRUE if you want `$Contents` to return a STRING for HtmlList objects.
- `SetActiveXBrowserStateActive` – default is FALSE. Set this option to TRUE, if SilkTest does not recognize the properties and methods of an ActiveX control in the browser. Setting the option to TRUE forces the DOM extension to behave as if the browser is ready when recognizing the ActiveX control, even though the browser 'Document complete' message was not received.
- `ScrollListItemIntoView` – default is TRUE. Set this option to FALSE to avoid scrolling a list box item or PopupList item into view before selecting it.
- `ShowHTCViewlink` – default is FALSE. Set this option to TRUE to allow the DOM extension to look for HTC ViewLinks. Setting this option to TRUE slows performance for recording and playback.
- `UseDocumentEvents` – default is FALSE. For more information, see HtmlPopupList causes the browser to crash when using IE DOM extension.
- `XMLNodeNamingVersion` – default is 0. For more information, see XMLNode class.

Options that you can set in domex.ini and via SetUserOption

To set the following options, you can edit domex.ini or you can use SetUserOption():

- `IgnoreSpanTags` – default is FALSE. If SilkTest recognizes multiple text objects as one HtmlText object and the object is a SPAN object that is parented to a SPAN object, set this option to TRUE to ignore the <SPAN> and <\SPAN> tags.
- `RowTextIncludesEmptyCells` controls the recognition of blank cells in bordered and borderless tables. This option is set to FALSE by default. If you want to return blank cells in tables as empty strings, set this option to TRUE.
- `ShowBodyText` – default is FALSE meaning that the DOM extension does not record BodyText objects, which are text that is not contained within an HTML tag. (In previous releases body text appeared as HtmlText.) Set this option to TRUE if you do want the DOM extension to record BodyText objects. We suggest keeping this option set to FALSE for improved performance, particularly when recording window declarations on large pages. You can also set this option on the DOM Extensions dialog.
- `ShowBorderlessTableFlags` indicates input elements that you do not want SilkTest to consider as input elements; for details, see Overview of input elements and borderless tables.
- `ShowBorderlessTables` – default is .5 meaning that the DOM extension does record BorderlessTable objects. However, .76 is the threshold where SilkTest starts to recognize more objects

within tables, such as images, hidden text, check boxes, textfields, and buttons. Set this option to .76 or greater if you want the DOM extension to record BorderlessTable objects. You can also set this option on the DOM Extensions dialog.

- `ShowHtmlForm` – default is FALSE meaning that the DOM extension does not record Form objects. Set this option to TRUE if you do want the DOM extension to record Form objects. You can also set this option on the DOM Extensions dialog.
- `ShowInvisible` – default is TRUE, meaning that invisible objects are recorded. This option lets you control whether or not invisible objects are recorded by the DOM extension. If your browser-based application consists of pages that contain many invisible objects that you do not need to test, then you can improve performance by setting the option to FALSE in order to ignore all invisible objects. You can also set this option on the DOM Extensions dialog.
- `ShowHtmlHidden` – default is TRUE meaning that the DOM extension records Hidden objects. Set this option to FALSE if you do not want the DOM extension to record Hidden objects. You can also set this option on the DOM Extensions dialog.
- `ShowHtmlMeta` – default is TRUE meaning that the DOM extension records Meta objects. Set this option to FALSE if you do not want the DOM extension to record Meta objects. You can also set this option on the DOM Extensions dialog.
- `ShowHtmlTable` – default is TRUE meaning that the DOM extension records HtmlTable objects. Set this option to FALSE if you do not want the DOM extension to record HtmlTable objects. You can also set this option on the DOM Extensions dialog.
- `ShowHtmlText` – default is TRUE meaning that the DOM extension records Text objects. Set this option to FALSE if you do not want the DOM extension to record Text objects. You can also set this option on the DOM Extensions dialog. If you are testing a transaction type page with lots of text consider not recording Text objects. This prevents SilkTest from recording the many text objects, which helps your declarations to be clean. If, on the other hand, you're looking for formatting and styles of text objects, you'll want to select this option.
- `ShowListItem` – default is TRUE. Set this option to FALSE if you do not want to show the text contained within HtmlList controls in your browser. If mouse events are associated with your list items, set this option to TRUE so SilkTest can interact with the list items. When set in the domex.ini file or DOM Extensions dialog, this setting is global. However, if you want to set this option for only certain points in your script, use BrowserPage.SetUserOption() as described in SetUserOption().
- `ShowOverflow` – default is TRUE meaning that SilkTest recognizes elements with overflow styles. These elements are very similar to IFrame elements in that they have their own scrollbar and can contain their own elements. Set this option to FALSE if you want SilkTest to ignore elements with overflow styles; this means that SilkTest may not interact with the elements contained by this overflow element.
- `ShowVirtualColumns` – default is FALSE. Affects how the DOM extension records asymmetric tables. These are tables that use either column span or row span attributes, or tables whose rows don't have the same number of columns. An example of an asymmetric table is a typical calendar page that has the month of January written across the top row and the seven days of the week in seven columns across the 2nd row. We recommend that you check this box if you are working with tables that have asymmetrical rows. Select this check box if you want SilkTest to create virtual columns for any row in a table. In the example below, it causes the top row to contain one real column for `January`, followed by 6 virtual columns which are blank. These virtual columns appear where there are none in order to complete the table and they are named virtual1, virtual2, and so on. These virtual columns cause the table to be symmetrical. Clear this check box to avoid creating virtual columns. This causes SilkTest to record the top row as the name for first column. This occurs because there is no 2nd column in the top row; Mon is promoted to the name of the second column, and so on. You can also set this option on the DOM Extensions dialog.

  ```
  January
  Sun Mon Tues Weds Thurs Fri Sat
  ```

- `SearchWholeDOMTree` – default is FALSE. This check box determines how windows declarations are found. If this value is set to TRUE, when recording window declarations, the search algorithm of current objects searches the whole DOM tree.

- `ShowXML` – default is TRUE meaning that the DOM extension records XML objects. Set this option to FALSE if you do not want the DOM extension to record XML objects. You can also set this option on the DOM Extensions dialog.
- `UseBrowserClosestText` – default is FALSE. Determines how the DOM extension finds the closest static text for HtmlTable, HtmlLink-text, HtmlColumn, HtmlLink-image, HtmlImage, HtmlHeading, HtmlText, HtmlRadioList, and HtmlPushbutton. Select this check box if you want SilkTest to use the DOM extension to find the closest static text for the objects listed above. This does not apply to invisible objects such as XML, Meta, and Hidden; those objects do not rely on any text on a page and so it would be meaningless to try to associate them with any objects. Clear this check box if you want to use the Agent to determine closest static text for the objects listed above. You can also set this option on the DOM Extensions dialog.
- `UseOverflowScrolling` – default is FALSE. Set this option to TRUE if off-screen HTML elements with overflow do not scroll into view properly.
- `UseScrollIntoView` – default is FALSE. This option is useful if you are having problems scrolling objects into view. This sometimes happens on HTML pages that contain scrollable iframes or scrollable HTCs. In general, the action/testcase recorder does not record the content inside an HTC. For other nested scrollable objects such as IFrames and overflow elements, the flashing rectangle appears in the wrong place, but the action/testcase recorder does generate correct script actions. If you are having problems playing back actions against nested scrollable objects, then set UseScrollIntoView to TRUE. Setting this option to TRUE helps avoid getting nested scrollable objects into view. Note that API-based clicking is not affected by the value of this option.

User options for table recognition

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following user options control table recognition:

RowTextIncludesEmptyCells	Controls the recognition of blank cells in bordered and borderless tables. This option is set to FALSE by default. If you want to return blank cells in tables as empty strings, set : `BrowserPage.SetUserOption ("RowTextIncludesEmptyCells", TRUE)`.
ShowBorderlessTables	Changes the level of recognition of tables in your web pages. See *Setting options for ShowBorderlessTables*.
ShowBorderlessTableFlags	Indicates input elements that you do not want to consider as input elements. See *Overview of input elements and borderless tables*.

Web Testing Methodology

Testing Methodology for Web Applications

You test Web applications with SilkTest using the same methodology as when you test standalone and client/server applications. Testing of both web-based applications and non-web-based applications includes these test phases:

- Creating and working with testplans.
- Designing and recording testcases.
- Running tests and interpreting results.
- Debugging testcases.
- Generalizing testcases.
- Handling exceptions.
- Making testcases data-driven.
- Customizing SilkTest.

Testing Web applications on different browsers

One of the challenges of testing web applications is that your users will probably be using different browser types and your application must support the browsers that your users use. When you develop tests for web applications running on different browser types, you must decide:

- How your testcases will handle differences between browsers.
- How to specify which browser to use for the testcase or test script.

Handling differences between browsers

In most cases, your include files (declarations) and scripts apply to any browser. You can run testcases against different browsers by simply changing the default browser and running the testcase, even if the pages look a bit different, such as pushbuttons being in different places. Because SilkTest is object-based, it doesn't care about layout. It just cares what objects are on the page.

There may be times when declarations and scripts have one or more lines that apply only to particular browser(s). In these situations you can use `browser specifiers` to make lines specific to one or more browsers. Browser specifiers are of the built-in data type `BROWSERTYPE`.

Testing Java applets and Java applications

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Java is a full-featured object-oriented language that can be used to make applets or full applications. A Java applet is embedded within a web page. A Java application is a complete standalone application.

The Java extension to SilkTest allows you to test Java applets and applications in various runtime environments.

For a complete description of the Java support, see *Overview of Java support*. You can also access a tutorial on testing Java applications and applets by clicking **Help** > **Tutorials**.

VO Automation

Information for current customers

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest no longer supports the VO extension and you must convert your automation to use the DOM extension. If you simply enable the DOM extension, there is no guarantee that previously written scripts will run without modification.

Using the DOM extension will require you to make changes to your scripts and include files. The number of automation changes will depend on how you have structured your tests and is hard to predict, since everyone has their own style of structuring their automation. See Changing existing VO automation to the DOM extension for more information.

There are differences in the way DOM and VO recognize objects and so you will have to make script changes regardless of which way you choose to use the DOM. For example, if you use the DOM extension, the caption tag as well as the window ID tag might be different for some objects than if you used VO. This means you would have to at least update your window declarations and possibly update your scripts.

Changing existing VO automation to the DOM extension

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

These are the steps you do to port your current VO automation to the DOM extension.

1. Make sure the DOM extension is on.
2. Record a new window declaration for the particular objects with which you are having difficulties.
3. Look for any differences in the identifier names and object hierarchy. Make changes in the include file and scripts where appropriate.

If you decide to use the DOM exclusively, use one of these two approaches after you turn on the DOM extension:

- Approach #1: Run your scripts and look for a `"Window '[class] Tagname' was not found"` error message. Look for any differences in the identifier names and object hierarchy. Repeat this until all errors are resolved.

- Approach #2: Re-record all of your declarations. Look for any differences in the identifier names and object hierarchy. Make changes in the include file and scripts where appropriate. This approach could potentially be more work than is necessary -- exactly how much work depends on how you have structured your automation.

Testing Objects in a Web Page

Internet Explorer Document Object Model Extension

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest uses the Document Object Model (DOM) extension which uses information in the HTML source to recognize and manipulate objects on a Web page.

Advantages of DOM

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The DOM extension has several advantages over VO:

- By default, when you are using the DOM extension the recorder displays a rectangle which highlights the controls as you are recording them.
- It is more accurate in certain situations, since it gets information directly from the browser. For example, the DOM extension recognizes text size and the actual name of objects.
- It is independent of the browser size and text size settings.
- It will find non-GUI (non-visible) objects such as `HtmlMeta`, `HtmlHidden`, `XML Node`, and `HtmlForm`.
- It offers better support for borderless tables.
- It is consistent with the standard being developed by the W3C.

In general, the DOM extension provides more classes and properties than are available with VO.

Additional information about DOM

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

This section contains notes about working with browsers

Internet Explorer

- When you use the DOM extension with IE, in order to interact with a browser dialog, the dialog must be the active (foreground) window. If another application is active, then SilkTest is not able to interact with the browser dialog, and the testcase times out with an "Application not ready" exception.
- You may receive a "Window not found error" when running scripts using the IE DOM extension. This error occurs when the testcase calls Exists() on the browser page before it is finished loading. This problem is due to the fact that the DOM extension does not check for DOM Ready in the Exists() method. The workaround is to call Browser.WaitForReady() in your script, prior to the Exists() method.
- For more information about how SilkTest recognizes tags, see the GetProperty method and GetTextProp method.
- You may see differences in image tags based on the same URL if you used two different URLs to get there. For example, SilkTest cannot differentiate between two images if IE displays two different URLs that both point to the same image.
- The DOM extension does not record inside a secure frame. This means that if an Html page contains frames with security (for example, on a banking page), the DOM extension on Internet Explorer will not be able to record the window declaration for the page because the secure site prevents DOM from getting any information.

Netscape Navigator and Firefox

There are several things to remember when you work with Netscape or Firefox and XUL. XUL (XML User-interface Language) is a cross-platform language for describing user interfaces of applications. SilkTest's support of XUL elements is limited. All menu, toolbar, scrollbar, status bar and most dialog boxes are XUL

elements. Almost all elements in the browser are XUL elements except the area that actually renders HTML pages.

- You can successfully record window declarations on the menu and toolbar by pointing the cursor to the caption of the browser. You can record actions and testcases against the menu and toolbar through mouse actions.
- You can also can record window declarations on a single frame XUL dialog box, such as the authentication dialog box. However, you cannot record window declarations on a multi-framed XUL dialog box, for example, the preference dialog box.
- SilkTest does not support:
- keyboard recording on the menu and toolbar (there is no keyboard recording on the URL)
- record actions and record testcase on XUL dialog boxes
- record identifier and location on XUL elements

See the Release Notes for more information about SilkTest's support of Netscape Navigator and Firefox.

Recording and playback

- When recording using the Internet Explorer DOM extension, a rectangle will flash to indicate the current target object of the recorder. In order to see the flashing rectangle when using the Netscape DOM extension, you must set the environment variable: `QAP_RECORD_FLASH=1`.
- The DOM extensions for Internet Explorer and Netscape are compatible, allowing you to develop (record or script) test scripts on one browser and play them back on the other browser with very little modification. Window declarations captured with the DOM extension in IE and Netscape are almost always identical. However, tags for HtmlText windows derived from body text may differ between browsers. Also, windows that use Agent closest static text for their tags may show tag differences because rendering is slightly different between IE and Netscape. In such cases, you can use browser specifiers to accommodate a separate tag for each browser.
- SilkTest can recognize XML Node objects in your Web page if your Web page contains XML content.
- The DOM extension supports HTCs, including those implemented using the Viewlink feature.
- It's a limitation of DOM that it cannot see the location of any text that is a direct descendant of the `<body>` tag. `GetRect()` does not work for body text. For example, when you record window declarations over body text, you do not get any objects. This was implemented for HTML pages where no <p> tags or other text formatting tags preface displayed text.
- DOM is limited in that it cannot find an insertion on a multi-line text field.
- Images created with the `<input type="image">` tag are seen as HtmlPushButtons.
- If you open a font statement on a web page with several HtmlText fields and HtmlCheckboxes, but do not close it off, the DOM extension will not recognize anything beyond the first object. Closing off the font statement with a tag enables SilkTest to work correctly.
- The DOM extension is not designed to handle multiple links with the same file name. If you do have multiple links, be sure to use the full URL to identify links.
- For Html pages that do not have explicit titles and which load Active X applications, you may have to modify test frames that you previously recorded using the VO Extension before you can use them with the DOM extension. This is because the DOM extension tags the BrowserChild slightly differently. Alternatively you could record new declarations for the page.
- HtmlTextfield GetPosition() always returns the last position. There is no method in the DOM which allows SilkTest to get the cursor position in an HtmlTextField.
- If you record a window declaration over a table that has indented links the indentation is recorded as an additional HtmlText object.
- When recording with the DOM extension TypeKeys ("<Tab>") are not captured. Since the script refers to object to type in directly, it is not necessary to record this manual Tab. You can manually enter a TypeKeys ("<Tab>") into your script if you want to; it just is not recorded.
- For more information about SilkTest's rules for object recognition, see the Rules for Object Recognition document. To open the document, choose Start > Programs > Silk > SilkTest > Documentation > SilkTest Classic > SilkTest Object Recognition.

4Test language and the DOM extension

- The ForceReady method is useful in situations where SilkTest never receives a 'Document complete' message from the browser. Unless SilkTest receives the 'Document complete' message, SilkTest acts as if the browser is not ready and will raise an 'Application not ready' error.
- The DOM extension supports several additional classes: `HtmlForm`, `HtmlHidden`, `HtmlMarquee`, `HtmlMeta`, and `XMLNode`.
- `FlushCache`, a DOM-only method, causes SilkTest to reexamine the currently loaded page (BrowserChild) and get any new items as they are generated, such as popup menus. This method is very useful when you are recording dynamic objects that may not initially appear.

Dynamic tables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If columns have objects in them, such as links or controls, SilkTest declares each of the links and controls when you record window declarations for the column.

If your application dynamically builds tables, such that you do not know at runtime how many rows there will be and consequently do not know how many objects there will be, you should not declare individual objects in tables. You should remove their declarations from those that SilkTest creates when you declare a window.

You can use the `GetRowChildren` method to get a list at runtime of all children (controls and objects) in a specified row of a table or column.

How SilkTest declares HTML frames

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

HTML frames are multiple web pages displayed concurrently in a browser. Each HTML frame is an independent scrollable region. The HTML frames form a hierarchy defined by the main web page. The main web page lays out the regions and the web pages associated with these regions. In more complex HTML designs, frames may be further nested.

SilkTest recognizes each frame as a `BrowserChild` and nests the `BrowserChild` objects as defined by the HTML frame hierarchy. The main web page is the root object and is recognized as `BrowserChild` object which may contain children of type `BrowserChild`.

SilkTest declares each frame on a web page as a child of the main window. SilkTest derives the identifier and tag of `BrowserChild` object from the title element in the html source. If the title element does not exist, then SilkTest will use the main page's title plus an index to identify each `BrowserChild`.

By default, SilkTest derives the identifier and tag of an HTML frame from its caption. In turn, the caption of an HTML frame is derived from the first text contained in the frame, such as an `HtmlText` or `HtmlHeading` element. If there is no text in an HTML frame, SilkTest derives the identifier from the frame's Window ID and the tag from the frame's index.

Related Topics

- *Streamlining HTML frame declarations*

Testing columns and tables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Tables in web applications are of the 4Test class `HtmlTable`. An `HtmlTable` consists of two or more columns of class `HtmlColumn`.

The `HtmlTable` and `HtmlColumn` class each have several methods and properties that allow you to get information about the tables in your application.

Definition of a table

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

For SilkTest, the definition of a table in HTML is:

- A table with 2 or more rows (specified with the <tr> tag in the page source).
- Where at least 1 row has 2 or more columns (specified via the <td> tag in the page source).

A single <td> with a colspan >1 does not qualify as 2 or more columns.

If a table with insufficient dimensions is nested inside other tables, then its parent tables will not be seen as HtmlTables even if they themselves have sufficient dimensions.

If a table does not meet this definition, SilkTest does not recognize it as a table. For example, if a table is empty (meaning that it has no rows or columns) and you attempt to select a row by using: table.SelectRow (1, TRUE, FALSE) you will get an error message saying E_WINDOW_NOT_FOUND, when you might expect to see a message such as E_ROW_INDEX_INVALID instead.

Testing controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Web applications can contain check boxes, combo boxes, list boxes, popup lists, pushbuttons, and radio lists, just as standard applications do. These objects in web applications are of class HtmlCheckBox, HtmlComboBox, HtmlListBox, HtmlPopupList, HtmlPushButton, and HtmlRadioList, respectively. All these classes are derived from their respective standard class. For example, HtmlCheckBox is derived from CheckBox. So all the testing you can do with these controls in standard applications you can also do in Web applications.

The following code gets the list of items in the credit card list in the **Billing Information** page of the sample GMO application.

```
LIST OF STRING lsCards
lsCards = BillingPage.CreditCardList.GetContents ()
ListPrint (lsCards)

Result:
American Express
MasterCard
Visa
```

Testing images

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Images in your application are objects of type HtmlImage. You can verify the appearance of the image by using the **Bitmap** tab in the **Verify Window** dialog.

If an HtmlImage is an image map, that is, if it contains clickable regions, you can use the GetRegionList, MoveToRegion, and ClickRegion methods to test those regions.

Testing links

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Links in your application are objects of type HtmlLink. SilkTest provides several methods that let you get their text properties as well as the location to which they jump.

For example, the following code returns the definition for the HtmlLink on a sample home page.

```
STRING sJump
sJump = Acme.LetUsKnowLink.GetLocation ()
STRING sJump
Print (sJump)

Result:
mailto:support@acme.com
```

Testing text in web applications

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Straight text in a standard web application can be in the following classes:

- `HtmlHeading`
- `HtmlText`

SilkTest provides methods for getting the text and all its properties, such as color, font, size, and style.

There are also classes for text in Java applets and applications. See the online Help for testing Java applets and applications for more information about Java classes.

For example, the following code gets the copyright text on a sample Web page.

```
STRING sText
sText = Acme.Copyright.GetText ()
Print (sText)

Result:
Copyright © 2006 Acme Software, Inc. All rights reserved.
```

Tips on how SilkTest recognizes objects in browsers

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Here are some notes on how SilkTest recognizes objects in browsers with the Document Object Model (DOM) extension:

- DOM uses the name attribute for input elements as the object's window ID. This makes object recognition for input objects independent of the way those object appear in a browser.
- The way HTML headings and HTML text are found -- to be considered an HtmlHeading by DOM the text must be tagged with <H1> through <H6>. If the text is tagged with `<TH>(table header)`, the text will be identified as a header if it is in the first row, or as HtmlText otherwise. If the text is simply bold it is considered simply a row and `GetTextProp("$FontStyle")` will record `FS_BOLD`. If you have bold text, DOM does not interpret that text as headings.
- How GetText works. With the DOM extension, `GetText` returns the first line as is defined by line break characters, if any (e.g.
). Because the DOM extension does not offer a visual interpretation of browser content, `GetText` always returns the same value regardless of browser size, font size, or browser.
- When you use the DOM extension, SilkTest attempts to group HTML text objects into one 4Test text object. However, SilkTest will separate objects if it encounters
 tags. This means if you use
 tags within your HTML pages, SilkTest may record more text objects than you expect. This is because with the DOM extension, SilkTest considers text separated by
 tags as separate objects. For example:

```
<p>
Welcome
<br>
and Opening Remarks
</p>
```

You might expect this to be recorded as one object, but SilkTest records this as two.

- For images, the DOM extension records both an `HtmlImage` and an `HtmlLink`
- If you spawn an additional browser window, SilkTest sees the second browser as another `BrowserPage` which means that you have to set the window active before interacting with it. This will ensure that you are working with the correct browser window.
- With the DOM extension, SilkTest captures only the first text style within a paragraph and assumes that style applies to the whole paragraph.
- For more information about SilkTest's rules for object recognition see the Rules for Object Recognition document.

Testing Borderless Tables

Overview of input elements and borderless tables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can use the `ShowBorderlessTableFlags` option in `domex.ini` to indicate input elements which you do not want SilkTest to consider as input elements.

This feature is provided as a convenience to you, but it has not yet been thoroughly tested.

To describe the HTML input element, you must use the tag you'd use in HTML. For example, setting `ShowBorderlessTableFlags=img` indicates that a borderless table having at least 1 image input element, described by the `<img>` HTML tag, is considered to have NO input elements at all, even if other input elements such as push buttons are contained in the table. This flag is implemented using OR functionality. For example:

```
ShowBorderlessTableFlags=img|input
```

means that any HTML table with EITHER `<img>` tag(s) OR `<input …>` tag(s) is considered to have NO input elements. To be even more specific about the type of input element you want to describe, you can use the values that are permitted for the "type" attribute of the input tag in HTML. For example,

```
ShowBorderlessTableFlags=submit
```

means that the presence of the HTML construct `<input type=submit>`, which creates a submit button, causes SilkTest to consider a table having this tag as having NO input elements.

Note:

- Input elements are: `HtmlTextField`, `HtmlImage`, `HtmlPushButton`, `HtmlPopupList`, `HtmlRadioButton`,`HtmlCheckBox`, `HtmlListBox`, and `HtmlHidden`.
- These settings do not apply to bordered tables. All bordered tables are recognized as tables.
- Except for the value of 1, all tables must meet the basic definition of a table.
- If performance is a consideration for you, consider setting the value for borderless tables to zero or 1. That causes SilkTest to find either no tables or all tables and your scripts will run faster.
- If the value for `ShowBorderlessTables` is set to less than .75, the `ShowBorderlessTableFlags` option is set, and the value matches an element in any cell of a table, then SilkTest will show that table.
- If the value for `ShowBorderlessTables` is set to less than .75, the `ShowBorderlessTableFlags` option is not set, and a cell contains an input element (`IMG`, `SELECT`, `INPUT`, `BUTTON` AND `TEXTAREA`), then SilkTest will not show the table. It will ignore it.

Guidelines to recognizing borderless tables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest, by default, uses `ShowBorderlessTables=.5`. At this setting, a borderless table which contains one or more input elements will not be considered to be an HtmlTable by SilkTest. Additionally, at the .5 setting, if a borderless table containing an input element is part of a set of nested borderless tables, none of the tables that contain that table , regardless of their content, will be considered to be an `HtmlTable`.

To change the level of recognition of tables in your web pages, you must set the value of the `ShowBorderlessTables` option.

Below are two charts that describe the guidelines for recognizing borderless tables on a Web page in SilkTest.

Use the chart below if the borderless table in question does not contain input elements.

Does the table have nested tables?	Do any of the nested tables have input elements?	Level of nested tables	To recognize this table as an HtmlTable, set ShowBorderlessTables to:
No	N/A	N/A	0 < x < .30

Does the table have nested tables?	Do any of the nested tables have input elements?	Level of nested tables	To recognize this table as an HtmlTable, set ShowBorderlessTables to:
Yes	No	1	.30 < x <.60
Yes	No	2	.60 < x < .75
Yes	No	> 2	1
Yes	Yes	Irrelevant	1

Use the chart below if the borderless table in question does contain input elements.

Does the table have nested tables?	Do any of the nested tables have input elements?	Level of nested tables	To recognize this table as an HtmlTable, set ShowBorderlessTables to:
No	N/A	N/A	.75<=x<.9
Yes	No	1 or 2	.75<=x<.9
Yes	No	3	.91<=x<.99
Yes	No	> 3	1
Yes	Yes	Irrelevant	1

Notes

All values between the suggested ranges in the preceding tables are the same. For example, it does not make any difference if you use 0 or 0.28 as borderless table value. SilkTest will ignore all borderless HtmlTables if the level is set to less than 0.0001, if the level is set to greater than or equal to .75, SilkTest will show all borderless tables.

Levels of recognition for borderless tables

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

The following chart describes in general how the ShowBorderlessTable values affect the tables that SilkTest recognizes.

To recognize	Then set the value to
no borderless tables as 4Test HtmlTables	ShowBorderlessTables=0
all borderless tables, regardless of content or dimensions, as 4Test HtmlTables.	ShowBorderlessTables=1

If performance is a consideration for you, consider setting the value for borderless tables to 0 or 1. SilkTest will find no or all tables and your scripts will run faster.

Setting options for ShowBorderlessTables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To change the level of recognition of tables in your Web pages, you must set the value of the `ShowBorderlessTables` option. There are two ways to do this.

- On the DOM Extensions dialog, check the **Table** check box and set a value for borderless table recognition. .76 is the threshold where SilkTest starts to recognize more objects within tables, such as images, hidden text, check boxes, textfields, and buttons.
- Tweak your script by setting the option within the script itself. This does not apply the value to the whole script but applies the value after that point in your script. You can use this method to adjust the level of recognition within your scripts, as you require. You do this by entering the following into your script: `BrowserPage.SetUserOption ("ShowBorderlessTables",value)`. For example, `BrowserPage.SetUserOption ("ShowBorderlessTables",.5)`.

Tag declaration for SSTab control

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If your application contains an `SSTab` control, which is associated with the class `OLESSTab`, you must use either the index or window ID for the tag in the window declaration. You cannot use the caption for the tag, because the caption changes based on which tab is selected.

Working with Borderless Tables

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Borderless tables are used to present content, but more commonly they are used to align text, figures, and other objects on a Web page. Often, borderless tables are nested within other borderless tables, many levels deep. Depending on the Web page, you may want to test only those tables that visually appear to be tables on the Web page, that is, those with actual borders. At other times, you might want to test a borderless table that presents content.

To meet your varying needs, the IE DOM extension has an option that allows you to set the level of recognition of tables. This is optional. You do not need to specify a value in order to use the DOM extension.

Testing XML

Testing XML

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can use SilkTest to verify that different renderings of your Web page display the same `XML` data. For example, if you change the presentation layer of your website, you can use SilkTest to "see" through the new presentation and test the `XML` data.

4Test Class

To support testing `XML` data, the 4Test language was modified to include the `XmlNode`. Users can access the `XML Elements` through properties and methods that have been defined within this new class.

Identifiers and tags with XML objects

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When creating identifiers and tags for XML objects, SilkTest first looks to the objects value. If no value is declared SilkTest takes the objects name/content. This is most important to remember in terms of Attributes. For example, the following is the source for some information about a book in a book catalog:

```
<Book BookType="Fiction" BookISBN ="x0682">
```

From this example we see that this book has two attributes: it is of BookType Fiction and its Book ISBN is x0682. The declaration for this appears as follows:

```
 [-] XmlNode Book1
  [+] multitag "Book [1]"
  [-] XmlNode FICTION
   [+] multitag "FICTION"
   [-]XmlNode X0682
    [+] multitag "x0682"
```

Note that SilkTest grabs the values Fiction and x0682 as the identifiers and tags.

Window declarations for XML

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When SilkTest captures the window declarations for an XML page, it first captures the presentation layer, the HTML objects that represent the XML data. Then below the declared HTML objects, the XML objects are declared.

Setting options for XML Recognition

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To set SilkTest to recognize XML elements

1. In SilkTest, choose **Options** > **Extensions** to display the **Extension** dialog.
2. Click in the **Primary Extension** column for the browser that you want to use, then select the browser name from the list box to enable the DOM extension.
3. In the **Options** area, click **Extension**.
4. On the **DOM Extension** dialog, check the **XML** check box and click **OK**.
5. Close the **Option Extensions** dialog, click **OK**.
6. Close the **Extensions** dialog, click **OK**.

To turn off XML recognition, uncheck the check box you checked in step 4 above.

Testing with the AOL Browser

Overview of SilkTest and the AOL Browser

SilkTest supports the AOL Browser. The AOL Browser extension is licensed and priced separately; for more information, contact Technical Support.

You can use the Basic Workflow to enable the AOL Browser extension only for the `waol.exe` executable. To use the extension with any other name for the AOL executable or DLL, you must enable extensions manually. To enable the extension manually for `waol.exe`, select the **AOL Browser** application and select **AOL Browser** as the Primary extension. For any other executable or DLL, use **New** to add the executable/DLL and select **AOL Browser** as the **Primary Extension**. See the list of extensions that SilkTest can automatically configure.

See the list of browser specifiers for more information about AOL browser types.

If you turn on the AOL browser in **Options** > **Extension**, SilkTest automatically loads the `extend \aol.inc` in the **useFiles** field in the **Options** > **Runtime** dialog. SilkTest also sets AOL as the default browser, if it is the only browser you have enabled.

Working with the AOL Browser

There are a few things to keep in mind while working with AOL.

- Create New Test Frame will record the `ChildWin` but not the `BrowserChild`. `DefaultBaseState()` will work properly as recorded. However, if you want to declare the `BrowserChild`, then you must record it using Record Window Declarations. See AOL BrowserChildren for more information
- AOL `BrowserChildren` are seen as children of a `ChildWin` whose parent is AOL. For more details, see AOL BrowserChildren.
- Automatic login is a feature of AOL and SilkTest assumes that you work with this feature turned on. For more information about AOL's Auto Login, see the AOL website. Because you need a unique username and password combination to work with AOL, the best way for SilkTest to work with this feature is to set the sPassword and sUsername properties in your testcase before you use them. For more information, see Specifying username and password.
- There is no support for AOL dialogs, only for common browser controls such as top level menus and pushbuttons.
- See the list of browser specifiers for more information about AOL browser types.
- Advertisement screens that appear after you log in may not be handled by default base state. These screens only appear once a day and must be manually closed by clicking "No Thanks".
- We've set SilkTest to ignore AOL toolbar and _AOL toolbar because these toolbars prevent the recognition of various buttons such as Stop. If you are working with option sets, you must include these three lines in your option set because option sets overwrite partner.ini. This is set up in partner.ini and controlled by the following three lines.

```
[Classmap]
_AOL_Toolbar=Ignore
AOL Toolbar=Ignore
```

AOL BrowserChildren

AOL `BrowserChildren` are now seen as children of a `ChildWin` whose parent is `AOL`. When recording window declarations, you can choose to declare the `BrowserChild` and its parent `ChildWin` separately. It is recommended that you first declare the `ChildWin`, so that when you record the `BrowserChild`, the 'parent' statement will be correct.

`BrowserPage` refers to the `BrowserChild` of the first AOL child window that contains a `BrowserChild`. That window may not be the active AOL child window, so you should call `BrowserPage.SetActive()` before using `BrowserPage`.

If you plan to make the window declaration compatible with the IE DOM extension, you should modify the `BrowserChild` declaration to include its `ChildWin` parent in its tag, so that its parent becomes 'Browser'. For example, if you record:

```
[+] window ChildWin MyAlerts

   [ ] tag "My Alerts"

   [ ] parent Browser

  [+] window BrowserChild MyAlerts2

    [ ] tag "My Alerts"

    [ ] parent MyAlerts
```

We recommend that you remove the `MyAlerts` declaration and modify the `MyAlerts2` declaration to:

```
[+] window BrowserChild MyAlerts

   [ ] tag "[ChildWin]*/[BrowserChild]My Alerts"

   [ ] parent Browser
```

Note that you can use a wildcard (*) for the caption of the `ChildWin`, or you can use the caption as recorded.

To make an AOL `BrowserChild` declaration work with IE, replace:

```
tag "[ChildWin]*/[BrowserChild]
```

with:

```
tag "{GetBrowserType() == AOL_IE ? "[ChildWin]*/[BrowserChild]" : ""}
```

To make an `IE BrowserChild` declaration work with AOL, preface the IE tag with:

```
"{GetBrowserType() == AOL_IE ? "[ChildWin]*/[BrowserChild]" : ""}
```

For example:

```
tag "{GetBrowserType() == AOL_IE ? "[ChildWin]*/[BrowserChild]" : ""}Acme
Software - Home"
```

However, the recorder cannot resolve expressions in tags. Therefore it is recommended that you use the cross-browser tag only for playback. When recording, comment out the cross-browser tag and uncomment the browser-specific tag.

For IE, the tag for this example would be:

```
tag "Acme Software - Home"
```

For AOL, the tag for this example would be:

```
tag "[ChildWin]*/[BrowserChild]Acme Software - Home"
```

Testing Client/Server Applications

ClientServer Application Support

SilkTest provides built-in support for testing client/server applications including:

- .NET WinForms
- Java AWT applications
- Java SWT/RCP application
- Java Swing applications
- Windows-based applications

In a client/server environment, SilkTest drives the client application by means of an Agent process running on each application's machine. The application then drives the server just as it always does. SilkTest is also capable of driving the GUI belonging to a server or of directly driving a server database by running scripts that submit SQL statements to the database. These methods of directly manipulating the server application are intended to support testing in which the client application drives the server. For additional information on this capability, see *Testing Databases*.

ClientServer Testing Challenges

SilkTest provides powerful support for testing client/server applications and databases in a networked environment. Testing multiple remote applications raises the level of complexity of QA engineering above that required for stand-alone application testing. Here are just a few of the testing methodology challenges raised by client/server testing:

- Managing simultaneous automatic regression tests on different configurations and platforms.
- Ensuring the reproducibility of client/server tests that modify a server database.
- Verifying the server operations of a client application independently, without relying on the application under test.
- Testing the concurrency features of a client/server application.
- Testing the intercommunication capabilities of networked applications.
- Closing down multiple failed applications and bringing them back to a particular base state (recovery control).
- Testing the functioning of the server application when driven at peak request rates and at maximum data rates (peak load and volume testing).
- Automated regression testing of multi-tier client/server architectures.

Verifying Tables in ClientServer Applications

This functionality is available only for projects or scripts that use the Classic Agent.

When verifying a table in a client/server application, that is, an object of the `Table` class or of a class derived from `Table`, you can verify the value of every cell in a specified range in the table using the **Table** tab in the **Verify Window** dialog box. For additional information on verifying tables in Web applications, see *Working with Borderless Tables*.

Specifying the range

You specify the range of cells to verify in the **Range** text boxes using the following syntax for the starting and ending cells in the range:

```
row_number : column_name
```

or

```
row_number : column_number
```

> **Example**
>
> Specifying the following in the **Range** text boxes of the **Verify Window** dialog box causes the value of every cell in rows 1 through 3 to be verified, starting with the column named ID and ending with the column named Company_Name:
>
> From field: 1 : id
>
> To field: 3 : company_name
>
> After you specify a cell range in the **Verify Window** dialog box, you can click **Update** to display the values in the specified range.

Specifying a file to store the values

You specify a file to store the current values of the selected range in the **Table File Name** text box.

What happens

When you dismiss the **Verify Window** dialog box and paste the code into your script, the following occurs:

- The values that are currently in the table's specified cell range are stored in the file named in the **Table File Name** text box in the **Verify Window** dialog box.
- A `VerifyFileRangeValue` method is pasted in your script that references the file and the cell range you specified.

For example, the following `VerifyFileRangeValue` method call would be recorded for the preceding example:

```
table.VerifyFileRangeValue ("file.tbl", {{"1",
"id"}, {"3", "company_name"}})
```

When you run your script, the values in the range specified in the second argument to `VerifyFileRangeValue` are compared to the values stored in the file referenced in the first argument to `VerifyFileRangeValue`.

For additional information, see the `VerifyFileRangeValue` method.

Evolving a Testing Strategy

There are several reasons for moving your QA program from local to remote testing:

- You may have a stand-alone application that runs on many different platforms and now you want to simultaneously drive testing on all the platforms from one SilkTest host system.
- You may have been testing a client/server application as a single local application and now you want to drive multiple instances of the application so as to apply a heavier load to the server.
- You may want to upgrade your client/server testing so that your testcases can automatically initialize the server and recover from server failures— in addition to driving multiple application instances.
- You may need to test applications that have different user interfaces and that communicate as peers.

If you are already a SilkTest user, you will find that your testing program can evolve in any of these directions while preserving large portions of your existing tests. This topic and related topics help you to evolve your testing strategy by showing the incremental steps you can take to move into remote testing.

Incremental Functional Test Design

SilkTest simplifies and automates the classic QA testing methodology in which testing proceeds from the simplest cases to the most complex. This incremental functional testing methodology applies equally well

in the client/ server environment, where testing scenarios typically proceed from the simplest functional testing of one instance of a client application, to functional and performance testing of a heavily loaded, multi-client configuration. Therefore, we recommend the following incremental progression for client/server testing:

- Perform functional testing on a single client application that is running on the same system as SilkTest, with the server application on the same system (if possible).
- Perform functional testing on a single remote client application, with the server application on a separate system.
- Perform functional and concurrency testing on two remote client applications.
- Perform stress testing on a single client application running locally or remotely.
- Perform volume load testing on a configuration large enough to stress the server application.
- Perform peak load testing on a large configuration, up to the limits of the server, if possible.
- Perform performance testing on several sets of loads until you can predict performance.

Network Testing Types

Software testing can be categorized according to the various broad testing goals that are the focus of the individual tests. At a conceptual level, the kinds of automated application testing you can perform using SilkTest in a networked environment are:

- Functional
- Configuration
- Concurrency

The ordering of this list conforms to the incremental functional testing methodology supported by SilkTest. Each stage of testing depends for its effectiveness on the successful completion of the previous stage. Functional, configuration, and concurrency testing are variations of regression testing, which is a prerequisite for any type of load testing. You can use SilkPerformer for load testing, stress testing, and performance testing.

You can perform functional testing with a single client machine. You can perform the first four types of test with a testbed containing only two clients. The last two testing types require a heavy multi-user load and so need a larger testbed.

How 4Test Handles Script Deadlock

It is possible for a multi-threaded 4Test script to reach a state in which competing threads block one another, so that the script cannot continue. This is called a script deadlock. When the 4Test runtime environment detects a deadlock, it raises an exception and halts the deadlocked script.

Example

The following script will never exit successfully.

```
share INTEGER iIndex1 = 0
share INTEGER iIndex2 = 0

main ()
  parallel
    access iIndex1
      Sleep (1)
        access iIndex2
          Print ("Accessed iIndex1 and iIndex2")
    access iIndex2
      Sleep (1)
        access iIndex1
          Print ("Accessed iIndex2 and iIndex1")
```

Troubleshooting Configuration Test Failures

The test of your application may have failed for one of the reasons below. If the following suggestions do not address the problem, you can enable your Extension manually. See *Enabling extensions*.

Note: Unsupported and embedded browsers, other than AOL, are recognized as client/server applications.

The application may not have been ready to test

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. Close and restart your application. Make sure the application has finished loading, and then click **Test**.

Embedded browsers, other than AOL, are recognized as Client/Server applications

If you want to work with a web browser control embedded within an application, you must enable the extension manually. See *Enabling embedded browsers*.

Types of Testing

Concurrency Testing

Concurrency testing tests two clients using the same server. This is a variation of functional testing that verifies that the server can properly handle simultaneous requests from two clients. The simplest form of concurrency testing verifies that two clients can make multiple non-conflicting server requests during the same period of time. This is a very basic sanity test for a client/server application.

To test for problems with concurrent access to the same database record, you need to write specific scripts that synchronize two clients to make requests of the same records in your server's databases at the same time. Your goal is to encounter faulty read/write locks, software deadlocks, or other concurrency problems. For an example of a test that demonstrates the necessary 4Test coding techniques, see *Code for Concurrency Test Example*.

Once the application passes the functional tests, you can test the boundary conditions that might be reached by large numbers of transactions.

Configuration Testing

A client/server application typically runs on multiple different platforms and utilizes a server that runs on one or more different platforms. A complete testing program needs to verify that every possible client platform can operate with every possible server platform. This implies the following combinations of tests:

- Test the client application and the server application when they are running on the same machine—if that is a valid operational mode for the application. This testing must be repeated for each platform that can execute in that mode.
- Test with the client and server on separate machines. This testing should be repeated for all different platform combinations of server and client.

Functional Testing

Before you test the multi-user aspects of a client/server application, you should verify the functional operation of a single instance of the application. This is the same kind of testing that you would do for a non-distributed application.

Once you have written scripts to test all the operations of the application as it runs on one platform, you can modify the scripts as needed for all other platforms on which the application runs. Testing multiple platforms thus becomes almost trivial. Moreover, many of the tests you script for functional testing can

become the basis of your other types of testing. For example, you can easily modify the functional tests (or a subset of them) to use in load testing.

Peak Load Testing

Peak load testing is placing a load on the server for a short time to emulate the heaviest demand that would be generated at peak user times—for example, credit card verification between noon and 1 PM on Christmas Eve. This type of test requires a significant number of client systems. If you submit complex transactions to the server from each client in your test network, using minimal user setup, you can emulate the typical load of a much larger number of clients.

Your testbed may not have sufficient machines to place a heavy load on your server system — even if your clients are submitting requests at top speed. In this case it may be worthwhile to reconfigure your equipment so that your server is less powerful. An inadequate server configuration should enable you to test the server's management of peak server conditions.

Volume Testing

Volume testing is placing a heavy load on the server, with a high volume of data transfers, for 24 to 48 hours. One way to implement this is to use one set of clients to generate large amounts of new data and another set to verify the data, and to delete data to keep the size of the database at an appropriate level. In such a case, you need to synchronize the verification scripts to wait for the generation scripts. The 4Test script language makes this easy. Usually, you would need a very large test set to drive this type of server load, but if you under-configure your server you will be able to test the sections of the software that handle the outer limits of data capacity.

Testing Java AWT/Swing Applications

Overview of Java AWTSwing Applications

SilkTest provides built-in support for testing applications and applets that use Java AWT/Swing controls. When you configure a Java AWT/Swing application or applet, SilkTest automatically provides support for testing standard Java AWT/Swing controls. You can also test Java SWT controls embedded in Java AWT/Swing applications or applets as well as Java AWT/Swing controls embedded in Java SWT applications.

SilkTest provides sample Java AWT/Swing test applications for both Open Agent and Classic Agent. You can download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest*. After you have installed the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java AWT** or **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java Swing** and then select the sample application that is right for your environment.

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

Object Recognition

Java AWT/Swing applications support hierarchical object recognition and dynamic object recognition. You can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements.

When you record a testcase with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Using custom class attributes becomes even more powerful when it is used in combination with dynamic object recognition.

To test Java AWT/Swing applications using hierarchical object recognition, record a test for the application that you want to test. Then, replay the tests at your convenience.

Supported Controls

For a complete list of the record and replay controls available for Java AWT/Swing testing with the Open Agent, refer to the Java AWT and Swing Class Reference in SilkTest's 4Test Language section of the Help.

For a complete list of the record and replay controls available for Java AWT/Swing testing with the Classic Agent, refer to the Java AWT Classes for the Classic Agent and Java JFC Classes in SilkTest's 4Test Language section of the Help.

SilkTest Agent Support

You can test Java AWT/Swing applications using the Classic Agent or the Open Agent. When you create a new Java AWT/Swing project, SilkTest uses the Open Agent by default. However, you can use both the Open Agent and the Classic Agent within a single Java AWT/Swing environment. Certain functions and methods run on the Classic Agent only. As a result, if you are running an Open Agent project, the Classic Agent may also open because a function or method requires the Classic Agent.

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

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

Classic Agent	Open Agent
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>
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>
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

Classic Agent	Open Agent
JavaAwtPopupList	AWTChoice
JavaAwtPopupMenu	<no corresponding class>
JavaAwtPushButton	AWTPushButton
JavaAwtRadioButton	AWTRadioButton
JavaAwtRadioList	<no corresponding class>
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*.

Java AWT Classes for the Classic Agent

Java AWT and Swing Class Reference

When you configure a Java AWT/Swing application, SilkTest Classic automatically provides built-in support for testing standard Java AWT/Swing controls.

Using the Open Agent

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.
7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.

8. Click Finish.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

    SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

    SilkTest opens the include file.
11. Record the test case whenever you are ready.

Configuring a Test Application that Uses the Java Network Launching Protocol (JNLP)

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

1. Record a test case for the application that you want to test.
2. In the INC file, replace the const `sCmdLine =` value with the command line pattern that includes the absolute path to `javaws.exe` and the URL to the Web Start.

   For example, to use the SwingSet3 JNLP application, type `const sCmdLine = "%ProgramFiles%\Java\jre6\bin\javaws.exe http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp"`

   When you replay the test case, the JNLP application starts as expected.

Custom Attributes

Add custom attributes to a test application to make a test more stable. You can use custom attributes with the following technologies:

- Java SWT
- Swing
- WPF
- xBrowser
- Windows Forms
- Win32
- SAP

For example, in Java SWT, the developer implementing the GUI can define an attribute (for example, `silkTestAutomationId`) for a widget that uniquely identifies the widget in the application. A tester using SilkTest can then add that attribute to the list of custom attributes (in this case, `silkTestAutomationId`), and can identify controls by that unique ID. Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index will change whenever another widget is added before the one you have defined already.

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

To create tests that use custom class attributes, you must use the Open Agent. For the following technologies, you can include custom attributes in tests that use hierarchical or dynamic object recognition:

- Java SWT
- Swing
- WPF
- Windows Forms
- Win32
- SAP

Because xBrowser only supports tests that use dynamic object recognition, you can only create tests that use dynamic object recognition. First, enable custom attributes for your application and then create the test.

Recording tests that use dynamic object recognition

Using custom class attributes becomes even more powerful when it is used in combination with dynamic object recognition. For example, If you create a button in the application that you want to test using the following code:

```
Button myButton = Button(parent, SWT.NONE);
myButton.setData("SilkTestAutomationId", "myButtonId");
```

To add the attribute to your XPath query string in your testcase, you can use the following query:

```
Window button = Desktop.Find(".//
PushButton[@SilkTestAutomationId='myButton']")
```

Recording tests that use hierarchical object recognition

This method is supported for Java SWT only.

When you record a test that contains a custom attribute, SilkTest records the custom attribute and includes the information in the frame.inc file. For example, the Java SWT sample application contains a custom attribute, `silkTestAutomationId` that is defined for the top-level window. When you record a testcase of the main window using `silkTestAutomationId` as the class attribute, the frame file contains the following:

```
window Shell SwtTestApplication
"&SilkTestAutomationId=shSWTTestApp"
```

Attributes for Java AWTSwing 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.

The attributes that SilkTest supports for Java AWT/Swing include:

- caption
- name
- accessibleName
- 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.)
- Swing only: All custom object definition attributes set in the widget with `SetClientProperty`("propertyName", "propertyValue").

Attribute names are case sensitive.

Dynamically Invoking Java Methods

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

Call dynamic methods on objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a control, use the `GetDynamicMethodList()` method.

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

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty()` method. To retrieve a list of supported dynamic properties for a control, use the `GetPropertyList()` method.

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

All built-in SilkTest Classic types.	SilkTest Classic types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
Enum types.	Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the enum type, java.sql.ClientInfoStatus you can use the string values of REASON_UNKNOWN, REASON_UNKNOWN_PROPERTY, REASON_VALUE_INVALID, or REASON_VALUE_TRUNCATED.
Other controls.	Control parameters can be passed as WINDOW.

Example

A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

```
customControl.Invoke("Reset")
REAL sum = customControl.DynamicInvoke("Add",{1,2})
```

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.

Using the Classic Agent

Overview of Java Support

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest provides support for testing stand-alone Java applications developed using supported Java virtual machines and for testing Java applets using supported browsers. You must configure SilkTest Java support before using it.

Testing standard objects and custom controls

Any single Java application or applet may contain a mix of standard and custom Java objects. With Java support, you can test both types of visible Java objects in applications and applets that you develop using the Java Development Kit (JDK)/Java Runtime Environment (JRE).

Standard Java objects are often defined in class libraries. SilkTest Java support lets you record and play back tests against standard controls by providing 4Test definitions for many Java classes defined in the following class libraries:

- Abstract Windowing Toolkit (AWT)
- Java Foundation Class (JFC), which includes the Swing set of GUI components
- Standard Widget Toolkit (SWT)
- Symantec Visual Café Itools

You can use the `setName`("<desiredwindow ID>") method to create a window ID that SilkTest will detect. `setName`() is a method inherited from class `java.awt.Component`, so it should work for most, if not all, of the Java classes that SilkTest can detect.

By contrast, custom controls often use native properties and native methods written in Java. Increasingly, custom controls also take the form of JavaBeans, which are reusable platform-independent software components written in Java. Developers frequently design custom controls to achieve functionality that is not available in standard class libraries. You can test custom Java objects, including JavaBeans, using SilkTest Java support.

The SilkTest approach to testing custom Java objects is to give you direct access to their native methods and properties. A major advantage of this methodology is that it obviates the need to write your own native methods using the SilkTest Extension Kit.

The procedure for testing custom Java objects is simple: Record a class for the custom control, then save the class definition in an include file. The class definition includes the native methods you can call and native properties you can verify from your 4Test script.

The predefined property sets supplied with SilkTest have not been customized for Java; however, you can modify these property sets. For additional information about editing existing property sets or creating new property sets, see *Creating a Property Set*.

Recording and playing back JFC menus

For Sun JDK v1.4 SilkTest can record and play back regular menus that conform to the Windows standard, as well as JFC heavyweight and lightweight pop-up menus.

Recording and playing back AWT menus

- Unlike JFC menus, AWT menus do not conform to the Java component-container paradigm. Therefore, their behavior is different than that of the JFC menus, and is independent of the JVM version. SilkTest can record regular AWT menus for all versions of the JDK.
- For context menus that conform to the windows standard, which means they can be opened with a right-click, SilkTest cannot record, but can play back for all versions of the JDK.
- For AWT popup menus that do not conform to the windows standard, SilkTest cannot record or play back for all versions of the JDK. `JavaAwtPopupMenu` is available for playback only. SilkTest is not able to record it and you must hand script any interaction with those menus.

Testing Java Applications and Applets

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest supports testing Java applications that use the Sun JDK. By default, SilkTest uses the Sun JDK with the SilkTest Classic Agent.

To prepare for testing stand-alone Java applications using SilkTest

1. In the **Basic Workflow** bar, enable extensions for Java support for application and applet testing.

   If you do not plan to test applets during the session, disable browser support.
2. Identify the custom controls in your Java application.
3. If you are testing Java applications, enable the recovery system.
4. Record classes for any custom controls you want to test in a new class include file or in your test frame file. If SilkTest does not recognize some of your custom objects, see recording classes for ignored Java objects.
5. If you are testing standalone Java applications, record window declarations for your Java application, including declarations for any new classes you defined.

Support for JavaBeans

This functionality is available only for projects or scripts that use the Classic Agent.

Many Java components are implemented as JavaBeans. Each JavaBean must provide an associated `BeanInfo` class that describes the capabilities of the component, including its methods and properties.

Our Java support provides a 4Test method called `GetBeanInfo` method that you can use in scripts to access information from the BeanInfo structure associated with JavaBeans in the applications or applets you are testing.

Using GetBeanInfo, you can access the following information about JavaBeans:

- Name of the JavaBean.
- Methods.
- Properties.
- Events supported by the JavaBean.
- Methods associated with event listeners supported by the JavaBean.
- Size of the icon associated with the JavaBean.

Accessing Sample Java applications and applets

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

We provide the following sample Java applications and applets:

Application Type	How to Access
Java application that uses AWT controls	Click **Start** > **Programs** > **<SilkTest program group>** > **Sample Applications** > **Java AWT** > **AWT Test Application**. Before working with this sample AWT application on your own, take the Java AWT quick tour in the SilkTest Tutorials.
Java Swing test application	Click **Start** > **Programs** > **<SilkTest program group>** > **Sample Applications** > **Java Swing** > **JFC Test Application**. Before working with this sample JFC Swing application on your own, see the SilkTest Tutorials.
Java AWT applet	Open `<SilkTest install directory>\JavaEx\jtapplet\index.htm`

You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*.

Select one of the following for instructions on running the sample applications or applet. You can also access Java tutorials by clicking SilkTest Tutorials.

- Running the sample applet in a browser
- Running the sample AWT application
- Running the sample JFC applications

SilkTest no longer ships with a JRE. In order to run the sample Java Swing application, you must open each file in a Text Editor and update the path to reflect the location of your local java.exe. If you do not have a local java.exe, you can download one from *java.sun.com*.

Classes in object oriented programming languages

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Classes are the core of object-oriented programming languages, such as Java. Applets or applications developed in Java are built around objects, which are reusable components of code that include methods and properties. Methods are tasks that can be performed on objects. Properties are characteristics of an object that you can access directly.

Each object is an instance of a class of objects. GUI objects in Java, for example, may belong to such classes as `Menu`, `Dialog`, and `Checkbox`. Each class defines the methods and properties for objects that are part of that class. For example, the `JavaAwtCheckBox` class defines the methods and properties for all Java Abstract Windowing Toolkit checkboxes. The methods and properties defined for `JavaAwtCheckboxes` work only on these checkboxes, not on other Java objects.

Configuring SilkTest to Test Java

Prerequisites for Testing Java Applications

This functionality is available only for projects or scripts that use the Classic Agent.

To test…	Install…
standalone Java applications	JDK/JRE
Java applets	JDK, supported browser, and plug-in (if necessary)
Java applets using the Java Applet Viewer	JDK and plug-in

Note:

- Java support is configured automatically when you use Enable Extensions in the basic workflow bar. You can use the basic workflow bar to configure your application or applet or manually configure Java Support. If you choose to manually configure Java support, you may need to

change the CLASSPATH environment variable. For JVM/JRE 1.2 or later, you must also copy the applicable SilkTest .jar file to the lib\ext folder of your JVM/JRE.

- SilkTest no longer ships with JRE 1.2. In order to run our sample Java Applications, you must open each *.bat file with a Text Editor and update the path to reflect the location of your local `java.exe`. If you do not have a local `java.exe`, you can download one from java.sun.com.
- The new version of the Java recorder does not support applets for embedded Internet Explorer browsers(AOL) or Netscape. These browsers use the previous version of the Java Recorder.

Overview of Enabling Java Support

This functionality is available only for projects or scripts that use the Classic Agent.

There are several ways to enable Java support for testing standalone Java applications. Pick the scenario that fits your runtime environment and testing needs.

Scenario	How to enable Java support
You need to test your application on the 32-bit Windows host machine using a JVM that is invoked from a java.exe, jre.exe, jrew.exe, or vcafe.exe executable, including: • JDK/JRE • Symantec Visual Café	Enable the default Java application.
You need to test your application on a remote 32-bit Windows machine using a JVM that is invoked from a java.exe, jre.exe, jrew.exe, or vcafe.exe executable.	Install SilkTest on your remote machine and enable the default Java application on your host machine.
You need to test your application on the 32-bit Windows host machine using a JVM that is not invoked from a java.exe, jre.exe, jrew.exe, or vcafe.exe executable.	Enable a new Java application.
You need to test your application on a remote 32-bit Windows machine using a JVM that is not invoked from a java.exe, jre.exe, jrew.exe, or vcafe.exe executable.	Install SilkTest on your target machine and enable a new Java application.

Configuring SilkTest Java Support for the Sun JDK

This functionality is available only for projects or scripts that use the Classic Agent.

Java support is configured automatically when you use **Enable Extensions** in the basic workflow bar. It is strongly recommended that you use the basic workflow bar to configure your application or applet, but it is also possible to manually configure Java support.

If you incorrectly alter files that are part of the JVM extension, such as the `accessibility.properties` file, in the `Java\lib` folder, or any of the files in the `jre\lib\ext` directory, such as `SilkTest_Java3.jar`, unpredictable behavior may occur. There are two methods for configuring SilkTest Java Support:

- Manually configuring SilkTest Java support.
- Configuring Standalone Java Applications and Java Applets.

Manually Configuring SilkTest Java Support

If you want to enable Java support manually, or if the Basic workflow does not support your configuration, click **Options** > **Extensions** to open the **Extensions** dialog box and enable Java applet or application support by checking or un-checking the **Java** check box for your application.

The **Java** check box can be checked or un-checked for a specific application or applet. If you check or un-check this check box for one extension, it is checked or un-checked for all.

Configuring Standalone Java Applications and Java Applets

In order for SilkTest to recognize Java controls, you may need to change the CLASSPATH environment variable. For JVM/JRE 1.3 or later, you must also copy the applicable `SilkTest .jar` file to the `lib`

\ext folder of your JVM/JRE. The SilkTest .jar file is located in the < SilkTest Install Directory>\JavaEx directory.

1. If you are using JVM/JRE 1.3 or later, use SilkTest_Java3.jar.

   For details about the supported Java versions, refer to the *Release Notes* in **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.
2. For Java 1.3 or later, you should not set a specific classpath variable – instead, use the default CLASSPATH=.;. Copy the SilkTest_Java3.jar file to the lib\ext folder of your JVM/JRE, and remove any previous SilkTest JAR files.
3. In the SilkTest folder, rename the file access3.prop to accessibility.properties and copy it to the Java…\lib folder.
4. Finally, qapjconn.dll and qapjarex.dll are new DLL files that must be installed in the Windows \System32 directory.

   The SilkTest installer places these files in the Windows\System32 folder, and also places copies of these files in the SilkTest\Extend folder. If the default directory for your library files is in a location other than Windows\System32, you must also copy qapjconn.dll and qapjarex.dll to the alternate location.

Note:

- The Java recorder does not support applets for embedded Internet Explorer browsers(AOL).
- It is not possible, using normal configuration methods, to gain recognition of Java applications that use .ini files to set the environment. However, if your application sets the Java library path using the JVM launcher directive Djava.library.path=< path >, you can obtain full recognition by copying qapjarex.dll and qapjconn.dll from the System32 directory into the location pointed to by the JVM launcher directive.
- If you are running your application from a .lax file, see *Configuring SilkTest to Support Java Application Launched from a .lax File*. If you are using JBuilder/JDeveloper, see *Configuring SilkTest for JBuilder/JDeveloper*.

Configuring SilkTest for JBuilderJDeveloper

This functionality is available only for projects or scripts that use the Classic Agent.

To test applications running in JBuilder or Oracle JDeveloper, perform the following steps:

Note: These steps refer to JBuilder; the same procedure applies to JDeveloper.

1. Determine which version of Java you are using and copy the SilkTest_Java3.jar file from <SilkTest installation directory>\extend to <JBuilder installation directory>\lib.
2. Edit the JBuilder.ini file to include the path and name of the SilkTest_Java3.jar file in the CLASSPATH.
3. Inside JBuilder, bring up the application you want to test.
4. Inside JBuilder, click **Project** > **Properties** and add the SilkTest_Java3.jar file to the list of libraries and call the library SilkTest.
5. Add the SilkTest library to the particular application you are testing.
6. Within SilkTest, make sure the Java extension in enabled. JBuilder uses the javaw.exe to run Java. Javaw.exe is part of the SilkTest Java extension.

Java Security Privileges Required by SilkTest

This functionality is available only for projects or scripts that use the Classic Agent.

Before reviewing your security privileges, make sure that you have configured SilkTest Java support.

Required security privileges

In order to load the SilkTest Java support files, SilkTest must have the appropriate Java security privileges. At a minimum, SilkTest requires the following abilities:

- Create a new thread.
- Access members of classes loaded by your application.
- Create, read from, and write to file on a local drive.
- Access, connect, listen and send information through sockets.
- Access AWT event queue.
- Access system properties.

For standalone applications, the security policy is set in the `java.security` file which is located in `JRE/lib/security`. By default this file contains the following line:

```
Policy.provider = sun.security.provider.PolicyFile
```

which means that the standard policy file should be used. The standard policy file, `java.policy`, is located in the same folder, `JRE/lib/security`. It contains the following code that gives all necessary permission to any file located in `lib\ext` directory:

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*"{permission
java.security.AllPermission;};
```

SilkTest has the necessary privileges, if the `SilkTest_Java3.jar` file is in this directory and the JVM runs with the default set of security permissions.

If you have changed the Java security policy

The system administrator can change security policy by starting the JVM with the following option:

```
java _Djava.security.policy=Myown.policy MyApp
```

In this case the custom policy file `Myown.policy` should contain the following lines that grant all permission to classes from the `lib\ext` directory:

```
grant codeBase "file:${java.home}/lib/ext/*"{permission
java.security.AllPermission;};
```

The default `java.policy` may also be changed implicitly, for example, when the application uses an RMI server with the custom RMISecurityManager and the client security policy. In cases like these, the client security policy should grant all required permissions to SilkTest by including the code listed above.

In some cases, setting these permissions may not provide SilkTest with the necessary security privileges. The cause of the problem may be that permissions are frame specific. So if SilkTest runs in the context of frames (thread) in which it does not have the necessary permissions, it may fail. In cases like this in which the client does not trust code running in the context of the AWT event thread, you need to set the parameter `ThreadSafe=False` in the `javaex.ini` in the `<SilkTest installation>/extend` directory. This prevents the SilkTest Java code from running in the context of the AWT event thread, preserving permissions granted to SilkTest, but could make the GUI less responsive.

Disabling Java Support

This functionality is available only for projects or scripts that use the Classic Agent.

To disable Java support:

1. Click **Options** > **Extensions** in the menu bar.
2. In the **Application** column, click the Java application that you want to disable and uncheck the **Java** check box, as follows:

If you installed ...	Uncheck Java check box for ...
Java Development Kit, Java Runtime Environment, or Symantec Visual Café	Java Application
Any other Java virtual machine	<name of executable file>.exe

3. Click **OK**.

Enabling Java Applications and Plugins

Enabling Extensions for the Default Java Application

This functionality is available only for projects or scripts that use the Classic Agent.

To enable the default Java application:

1. Click **Options** > **Extensions** in the menu bar.
2. In the **Application** column, click **Java Application** and check the **Java** check box.
3. Click **OK**.

Enabling a New Extension for a Java Application

This functionality is available only for projects or scripts that use the Classic Agent.

You must enable the new Java application in two locations: **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Extension Enabler** and then within SilkTest by choosing **Options** > **Extensions**. Any JVM other than what ships with Sun's JDK is an alternate JVM.

1. Start the Extension Enabler by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Extension Enabler**.

   If you are running your Java application on a remote 32-bit Windows machine, launch `extinst.exe`, which is in the SilkTest installation directory on the remote machine.
2. On the **Extension Enabler** dialog box, click **New**.
3. Click  to navigate to the location of the JVM executable you want to hook into, and then click **OK**.
4. Check the **Java** check box for the executable you just added, leave Primary Extension set to (None), and then click **OK**.
5. Start SilkTest and click **Options** > **Extensions**.
6. On the **Extensions** dialog box, click **New**.
7. Click  to navigate to the location of the JVM executable you want to hook into, and then click **OK**.
8. Check the **Java** check box for the executable you just added, leave Primary Extension set to (None), and then click **OK**.
9. Start your Java application.

Enabling Use of Sun Java Plug-In

This functionality is available only for projects or scripts that use the Classic Agent.

To use the JRE for running Java applets in any of our supported browsers, you must enable use of a plug-in and your applet must explicitly request to run in the Java plug-in.

To enable use of a plug-in:

1. Make sure you have installed the Java plug-in for each of the supported browsers you want to use for testing.
2. Click **Options** > **Extensions** in the menu bar.
3. In the **Application** column, click one of the browsers for which you installed the Java plug-in. If not enabled, then select **Enabled** from the list box in the **Primary Extension** column.
4. In the **Options** section, click **Extension**.

5. In the **Extension Options** dialog box, check the **Enable use of Java plug-in** check box, and then click **OK**.
6. Repeat steps 3–5 for other supported browsers.
7. Click **OK** to close the **Extensions** dialog box.

You can designate the JRE that the plug-in uses by clicking **Start** > **Settings** > **Control Panel** > **Java Plug-in** and selecting the JRE on the **Advanced** tab of the **Java Plug-in Properties** dialog box. Make sure that the appropriate SilkTest .jar file is accessible to the plug-in. For additional information, see *Configuring Stand-Alone Java Applications and Java Applets*.

Configuring SilkTest to Support a Java Application Launched from a .lax File

This functionality is available only for projects or scripts that use the Classic Agent.

If you are running your Java application from a launcher application executable (*.lax), you must add the appropriate SilkTest .jar file to the CLASSPATH inside the .lax file.

To find out which version of Java the .lax file is using, open a Command Prompt, type the path to Java that displays in the .lax file, and then type `java –version`. If it is using Java 1.3 and later, add the `SilkTest_Java3.jar` to the CLASSPATH inside the .lax file.

If you are running your application through a .lax file, make sure you add the .lax file as a new extension in SilkTest.

Testing Applications and Applets

Testing Java Applications and Applets

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest supports testing Java applications that use the Sun JDK. By default, SilkTest uses the Sun JDK with the SilkTest Classic Agent.

To prepare for testing stand-alone Java applications using SilkTest

1. In the **Basic Workflow** bar, enable extensions for Java support for application and applet testing.

   If you do not plan to test applets during the session, disable browser support.
2. Identify the custom controls in your Java application.
3. If you are testing Java applications, enable the recovery system.
4. Record classes for any custom controls you want to test in a new class include file or in your test frame file. If SilkTest does not recognize some of your custom objects, see recording classes for ignored Java objects.
5. If you are testing standalone Java applications, record window declarations for your Java application, including declarations for any new classes you defined.

Testing Browser-Based Java Applications

This functionality is available only for projects or scripts that use the Classic Agent.

The following procedure applies to JDK 1.4 applications:

1. Make sure the Java extension and appropriate browsers are enabled.
2. Make sure that SilkTest is configured properly.
3. Open the file `<SilkTest installation directory>\Silk\SilkTest\JavaEx\JFC \JPI_index.html` in the browser you want to use for testing.
4. Start SilkTest and record as usual.

If you are testing JDK 1.4 browser-based applications and the application throws security exceptions, make sure that the `SilkTest_Java3.jar` file was copied to the correct location. Installing security certificates in the browser does not resolve this issue; the `SilkTest_Java3.jar` must be in the correct location.

Indexed Values in Test Scripts

This functionality is available only for projects or scripts that use the Classic Agent.

4Test methods use a 1-based indexing scheme, where the first indexed value is stored in position 1. Native Java methods use a 0-based indexing scheme, where the first indexed value is stored in position 0. This incompatibility can create challenges in coding test scripts that access indexed values using both native methods and 4Test methods.

Multitags and Java Applications

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest permits multitags when recording window declarations for Java applications. You are no longer restricted to just the caption. The exception is that the default for top-level windows is only the caption. The reason is that the window ID usually defaults to the class name with an index, and for top-level windows, the index ("#1") leads to misidentification. If the window ID will be unique for a top-level window, then you can highlight that window in the **Record Window Declarations** dialog box and check **Window ID** in the **Tag Information** box.

When to Use 4Test Versus Native Java Controls

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest provides a predefined set of Java classes, including Abstract Windowing Toolkit (AWT) controls, Java Foundation Class (JFC) controls, and Symantec Visual Café controls. To test these controls, you can use their inherited 4Test methods. Inherited methods are the 4Test methods associated with the class from which the control is derived.

For custom Java controls, we provide access to native Java methods, as defined in JDK 1.1.2 or later. You can also access native methods for predefined Java classes.

When both 4Test methods and native methods are available for all controls you want to test, we recommend using 4Test methods in your test scripts. 4Test provides a richer, more efficient set of methods that more closely mirror user interaction with the GUI elements of an application. For the `JavaAwtPushButton`, for example, use the 4Test methods associated with the `PushButton` class.

When you must use native methods for controls that are not supported in 4Test, refer to the Java API documentation to gain a full understanding of how the native method works. For example, 4Test methods and native Java methods use incompatible array indexing schemes so you must use caution when accessing indexed values.

Web Applications and the Recovery System

This functionality is available only for projects or scripts that use the Classic Agent.

When the recovery system needs to restore the base state of a Web application that uses the Classic Agent, it does the following:

1. Invokes the default browser if it is not running.
2. Restores the browser if it is minimized.
3. Closes any open additional browser instances or message boxes.
4. Makes sure the browser is active and is not loading a page.
5. Sets up the browser as required by SilkTest Classic.

   The recovery system performs the next four steps only if the `wMainWindow` constant is set and points to the home page in your application.
6. If `bDefaultFont` is defined and set to TRUE for the home page, sets the fonts.
7. If `BrowserSize` is defined and set for the home page, sets the size of the browser window.
8. If `sLocation` is defined and set for the home page, loads the page specified by `sLocation`.
9. If `wMainWindow` defines a `BaseState` method, executes it.
10. For additional information, see *DefaultBaseState and the wMainWindow Object*.

To use the recovery system, you must have specified your default browser in the **Runtime Options** dialog box. If the default browser is not set, the recovery system is disabled. There is one exception to this rule: You can pass a browser specifier as the first argument to a test case. This sets the default browser at runtime. For more information, see *BROWSERTYPE Data Type*.

The constant `wMainWindow` must be defined and set to the identifier of the home page in the Web application for the recovery system to restore the browser to your application's main page. This window must be of class `BrowserChild`. When you record a test frame, the constant is automatically defined and set appropriately. If you want, you can also define a `BaseState` method for the window to execute additional code for the base state, for example if the home page has a form, you might want to reset the form in the `BaseState` method, so that it will be empty at your base state.

On Windows Internet Explorer 7.x and 8.x, when recording a new frame file using **Set Recovery System**, by default SilkTest Classic does not explicitly state that the parent of the window is a browser. To resolve this issue, add the "parent Browser" line to the frame file.

Java extension options

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

There are several options that you can set for the Java extension by manually editing the javex.ini file. These options apply to `SilkTest_Java3.jar` only.

The settings go in the `[Options]` section of `javaex.ini` unless otherwise noted. These are optional and you do not have to include the setting in your `javaex.ini` if you want the default behavior.

`AwtCompTreeLockTimeout`

Default is -1. This option applies only to AWT-based applications that contain AWT popup menus. It should not be used with other applications because it will slow performance. If your application contains AWT popup menus, then you should set this value to a positive number representing milliseconds (for example, 1000, representing 1 second) in order to prevent SilkTest from freezing when interacting with AWT popup menus.

It is strongly recommended that you use `PopupSelect()` rather than `JavaAwtPopupMenu::Pick()`.

`EnumAwtPeers`

Default is TRUE. This option applies only to AWT-based applications. It will not affect applications based on JFC (Swing). Setting the option to FALSE may significantly improve playback speed. However, it may change the window identifier hierarchy, which may cause existing automation to fail. The default value is TRUE, for backward compatibility with previous versions of SilkTest.

The option controls whether SilkTest searches for peers of AWT controls. Peers are windows that represent counterparts to Java controls, but are not Java classes themselves. Examples of peer classes that you may see in your window declarations are `SunAwtFrame` or `SunAwtCanvas`. If the option is set to FALSE, SilkTest does not recognize peer classes, and so does not include them in the window identifier hierarchy. You should not set the option to FALSE if any of your window declarations mention such classes; otherwise you will have to change your existing test scripts to accommodate the modified window identifier hierarchy.

If the option is set to FALSE, SilkTest will, as always, recognize most custom Java components without having to add their class names to the `[ClassList]` section of `javaex.ini`. There are two exceptions: custom components that have children, and custom components that are derived from ignored container classes such as `Panel`. In order to enable SilkTest to recognize a component in one of those two categories, you must add the class name to the `[ClassList]` section of `javaex.ini`. See *Recording classes for ignored Java objects* for more information. In order to identify the classes to add to the `[ClassList]` section, you may need to enable **Show all classes** on the **Record Class** dialog and examine the resulting window declarations. You must remember to disable **Show all classes** before you modify the `[ClassList]` section. Never play back scripts with **Show all classes** enabled, or performance will slow down greatly.

IncludeInstanceNumber

Default is FALSE. The Java extension uses the component name as the window ID form of the tag. For most JFC (Swing) components, the component name is the component class name. For AWT components, however, the component name has the form: base name + instance number. The base name resembles the component class name. For example, AWT Button has a base name of "button", while AWT Dialog has a base name of "dialog". The instance number reflects the order in which this instance of the component was created by the JVM. The order of creation, and therefore the instance number, may vary depending on the sequence of actions performed against the application. This may cause the window ID of a specific window to vary between runs of a testcase, or even within a testcase.

For example, the first time that you invoke a specific popup window, its window ID will be "dialog0" (base class = "dialog"; instance number = 0). But if you discard this popup window and then invoke it again, the window ID will be "dialog1". This will invalidate the window declaration if the window ID is the only tag form being used for that window. If `IncludeInstanceNumber` is set to FALSE (the default value), then the instance number will be omitted from the window ID, and the window ID will consist only of the base class. If there are multiple windows with the same base class, then the window ID's will have the form "baseClass[n]", following SilkTest's standard convention for distinguishing between multiple controls with the same tag value.

Setting the option to TRUE will reintroduce the Java instance number into the window ID tag form, which usually will make the window ID less robust. However, the option is included for purposes of backward compatibility.

ParentPopupToInvoker

Default is TRUE. This option specifies how the parent of a popup menu should be determined. When the option is set to TRUE (default), the parent of the popup menu will be the window that invoked it (or the first ancestor of the invoking window that is not ignored by the Java extension). When the option is set to FALSE, the parent of the popup menu will be the container of the popup menu (or the first ancestor of the container that is not ignored by the Java extension).

Recognition of popup menus should be more robust using the default value, TRUE, because that value will eliminate changes to the window hierarchy that occur when the container of a popup menu changes because the size or placement of the menu changes. Prior to the introduction of this option, the container was always used as the parent of the popup menu, corresponding to an option value of FALSE.

For example, for JFC (Swing) menus, the container by default is a panel. If the panel is ignored by the Java extension, as usually will be the case, and if SilkTest is using the container as the parent (option value is FALSE), then SilkTest will recognize the parent as the `JavaMainWin` that contains the menu. However, if the menu extends beyond the boundary of the `JavaMainWin`, then the container, and therefore the parent (if option value is FALSE) will be seen as a popup window (usually a `JavaDialog`). So if the option value is FALSE, then the parent of the menu may change depending on the size of the menu, causing the window declaration to be invalid. If SilkTest is using the invoking window as the parent (meaning that this option is TRUE), however, then the parent will not change because the same window (usually the `JavaMainWin`) invokes the menu whether or not the menu lies within the boundaries of the `JavaMainWin`.

The default value, TRUE, also allows SilkTest to distinguish between popup menus that are invoked in different contexts, for example by clicking different buttons in the same toolbar. The parent of the popup menu is the button that was clicked to bring up the menu. In contrast, setting the value to FALSE may cause popup menus invoked by clicking different buttons in a toolbar to be seen as children of the same `JavaMainWin`, and therefore to be seen as part of the same popup menu.

TableGetValueAtOnly

Default is FALSE. This option, which is part of the `[JavaJFCTable]` section of the `javaex.ini` file, determines how SilkTest obtains the cell text for `JavaJFCTable` controls. The default value, FALSE, uses the cell renderer object to find the cell text for non-string cell contents, such as for a cell that contains a custom component that displays text but does not implement `toString()`. If SilkTest does not return the

correct table cell text with the value set to FALSE, then change the value to TRUE. Setting the value to TRUE causes SilkTest to use the cell data object rather than the renderer object.

`TreeNodeValueHasPrecedence`

Default is FALSE. This option, which is a part of the `[JavaJFCTreeView]` section of the `javaex.ini` file, determines how SilkTest obtains the node text for `JavaJFCTreeView` controls. With this option set to FALSE, SilkTest attempts to find meaningful text for the tree node by relying more heavily on the Java API. If SilkTest does not return the correct tree node text with the value set to FALSE, then change the value to TRUE. Setting the value to TRUE gives precedence to the string representation of the value of the node, even if that value does not yield apparently meaningful text.

`UseExpandButton`

Default is FALSE. This option lets you specify how SilkTest should expand/collapse nodes in a `JavaJFCTreeView`. The default value, FALSE, specifies that SilkTest should double-click nodes to expand or collapse them. This value works for default implementations of a `JavaJFCTreeView`. However, if your implementation of a `JavaJFCTreeView` does not use a double-click to expand or collapse nodes, then set the value to TRUE, which directs SilkTest to click on the **Expand** button (usually a small square with a +/- sign).

Setting Java Extension Options Using the javaex.ini File

This functionality is available only for projects or scripts that use the Classic Agent.

You can set several options for the Java extension by manually editing the javaex.ini file. These options apply to `SilkTest_Java3.jar` only

To set Java extension options using the `javaex.ini` file:

1. Close SilkTest and the AUT, if they are open.
2. Open `javaex.ini`, located in the extend subdirectory of the directory where you installed SilkTest. If you are using a SilkTest Project, the applicable `javaex.ini` file is in the project, not in the SilkTest install directory.
3. Go to the `[Options]` section, unless otherwise noted, and change the value of the option. You may need to add a line containing the section name, if it does not already exist.
4. Save and close the `javaex.ini` file.
5. Restart SilkTest.

Troubleshooting Test Failure for Standalone Java Application

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The test of your standalone Java application may have failed for the reasons below. If the following suggestions do not address the problem you are having, you can enable your extension manually. See *Configuring SilkTest Java Support* and *Enabling extensions*.

The application may not have been ready to test

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. Click **OK** on the **Extension Settings** dialog box, and then close and restart your application.
4. Make sure the application has finished loading, and then click **Test** .

If you started your application via a Command Prompt, you may have forgotten to close and re-open the Command Prompt when you restarted your application

Using the **Basic workflow** bar, enable the extension again, making sure that you close and re-open both your Java application and the **Command Prompt** window before you click **Test** on the **Test Extension Settings** dialog box.

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the Java application for which you want to enable extensions, and then click **OK** on the **Extension Settings** dialog box.
3. Close your Java application and the **Command Prompt** window.
4. Open a **Command Prompt** and restart your application.
5. Make sure the application has finished loading, and then click **Test**. The Test should now pass.

Your Java application does not contain any Java children within JavaMainWin

The test will fail if your Java application (or applet) does not contain any Java children within JavaMainWin. However, the Java extension may be configured properly. Record against Java controls to make sure the extension is enabled. For example, a push button should be recorded as a `JavaAWTPushButton` or a `JavaJFCPushButton`.

Troubleshooting Test Failure for Applet Configuration

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The test of your applet may fail if the application was not ready to test, the Java plug-in was not enabled properly, if there is a Java recognition issue or if the Java applet does not contain any Java children within the **JavaMainWin**.

If the browser has a plug-in enabled or the applet uses a plug-in, you must select the **Java Plug-in** check box on the **Extension Settings** dialog box. In all other cases, make sure that the Java plug-in check box is unchecked.

If the following suggestions do not address the problem you are having, you can enable your extension manually.

The application may not have been ready to test

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. Close and restart your application. Make sure the application has finished loading, and then click **Test**.

Plug-Ins

If the browser has a plug-in enabled or the applet uses a plug-in, you must select the **Java Plug-in** check box on the **Extension Settings** dialog. In all other cases, make sure that the **Java plug-in** check box is unchecked.

To determine if Internet Explorer 6 has a plug-in enabled, click **Tools** > **Internet Options** > **Advanced** tab. Scan the **Settings** list to see if a third party plug-in, such as Java (Sun), has been enabled.

A plug-in is enabled for the browser, and the applet is using a plug-in, but you did not select the Java Plug-in check box

Using the **Basic workflow** bar, enable the extension again, making sure that you select the **Java Plug-in** check box on the **Extension Settings** dialog.

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. On the **Extension Settings** dialog box, make sure `DOM` is the **Primary Extension**, select the **Java Plug-in** check box, and then click **OK** .
4. Close and restart your application. Make sure the application has finished loading, and then click **Test**. The Test should now pass.

You are testing an applet that does not use a plug-in, but the browser has a plug-in loaded

1. Disable all plug-ins for the browser.

2. Enable the extension again, using the **Basic workflow** bar.
3. Make sure that the **Java Plug-in** check box is not selected on the **Extension Settings** dialog box.

The SilkTest_Java3.jar file is not in the lib/ext directory of the plug-in that you are using

Locate the `lib/ext` directory of the plug-in that you are using and make sure that the `SilkTest_Java3.jar` file appears in this folder.

If the file is not in this folder, copy the `SilkTest_Java3.jar` file from the `<SilkTest installation directory>\javaex` folder to the `lib\ext` directory of the plug-in that you are using.

SilkTest could not recognize any Java children within the applet

This error may occur if your applet contains only those Java classes that are not currently recognized, such as a frame containing only an image. See *Mapping a custom class* for information about mapping to recognized classes.

Make sure that the Java security privileges required by SilkTest are set. See *Java security privileges required by SilkTest* for information.

Classes Methods and Functions

Java Classes Supported

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

We provide 4Test definitions in our class definition file for the following Java classes:

- Abstract Windowing Toolkit (AWT) classes
- Java Foundation Class (JFC) library classes
- Symantec Visual Café Itools classes
- Java-equivalent window classes

Each of these predefined classes inherits 4Test properties and methods, which are referenced in the class descriptions in this Help. Not all inherited methods have been implemented for Java controls.

You can also access their native methods by removing the 4Test definition and re-recording the class.

The only assumption that the Java extension makes about implementation of Java classes in an AUT is that the classes do not violate the standard Swing or AWT models. The Java extension should recognize and be able to manipulate a Java class in an application as long as it extends one of the components that the Java extension supports and any customization does not violate that component's API. For example, changing a method from public to private violates the component's API.

Predefined AWT classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following 4Test classes are provided for testing Abstract Windowing Toolkit (AWT) controls:

- `JavaAwtCheckBox`
- `JavaAwtListBox`
- `JavaAwtPopupList`
- `JavaAwtPopupMenu`
- `JavaAwtPushButton`
- `JavaAwtRadioButton`
- `JavaAwtRadioList`
- `JavaAwtScrollBar`
- `JavaAwtStaticText`
- `JavaAwtTextField`

Predefined JFC classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following 4Test classes are provided for testing Java Foundation Class (JFC) controls:

- `JavaJFCCheckBox`
- `JavaJFCCheckBoxMenuItem`
- `JavaJFCChildWin`
- `JavaJFCComboBox`
- `JavaJFCImage`
- `JavaJFCLayeredView`
- `JavaJFCListBox`
- `JavaJFCMenu`
- `JavaJFCMenuItem`
- `JavaJFCPageList`
- `JavaJFCPopupList`
- `JavaJFCPopupMenu`
- `JavaJFCProgressBar`
- `JavaJFCPushButton`
- `JavaJFCRadioButton`
- `JavaJFCRadioButtonMenuItem`
- `JavaJFCRadioList`
- `JavaJFCScale`
- `JavaJFCScrollBar`
- `JavaJFCSeparator`
- `JavaJFCStaticText`
- `JavaJFCTable`
- `JavaJFCTextField`
- `JavaJFCToggleButton`
- `JavaJFCToolBar`
- `JavaJFCTreeView`
- `JavaJFCUpDown`

Predefined Symantec Itools classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following 4Test classes are provided for testing Symantec Visual Café Itools controls:

- `JavaItoolsComboBox`
- `JavaItoolsListBox`
- `JavaItoolsPageList`
- `JavaItoolsPushButton`
- `JavaItoolsScale`
- `JavaItoolsTable`
- `JavaItoolsTreeView`
- `JavaItoolsUpDown`

Predefined Java-equivalent window classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following 4Test classes are provided for testing Java-equivalent window controls:

- `JavaApplet`
- `JavaDialogBox`
- `JavaMainWin`

Recording Classes

When to Record Classes

This functionality is available only for projects or scripts that use the Classic Agent.

Consider these criteria when deciding whether to record classes for Java objects. When you record classes, SilkTest derives tags from the Java class name. You may also find it helpful to consult the decision tree for dealing with custom Java classes.

Am I testing only predefined Java classes?

If you are testing only predefined Java classes, then you do not need to record additional classes. Check the list of predefined Java classes to be sure. If you want to access native methods for predefined Java classes, then you must remove the existing definition and re-record the class.

Am I testing visible custom controls?

If you are testing custom Java controls that are not predefined, then you must record classes for these controls. In this case, the custom controls are visible, but display as `CustomWin` objects. After you record the class, you can retrieve information about any number of instances (objects) of that class.

Do I want to test custom controls that are currently ignored?

To maintain efficiency during the recording process, SilkTest ignores custom Java controls that are not considered relevant for testing, such as containers or panels. Ignored objects are not recognized at all by SilkTest, not even as `CustomWin` objects.

Even so, you can expose and record classes for ignored Java objects in standalone Java applications or in Java applets that you consider important for testing purposes.

Have I modified an existing class definition?

If you add, delete, or modify any native methods or parameters for a custom Java class, you need to either re-record the class or modify your class include file to reflect the changes.

Decision tree for dealing with custom Java classes

1. Can you see the object without **Show All Classes** checked?
    a. If yes, then do you get any methods and properties using **Record Class**?
        a. If yes, then use **Record Class** to generate a winclass for the custom class. We recommend that you check **Show all methods**.
        b. If no, then do you get any methods and properties using the `CaptureObjectClass`() or `CaptureAllClasses`() function?
            a. If yes, then use `CaptureObjectClass`() to generate a winclass for the custom class, or `CaptureAllClasses`() to generate winclasses for the custom class and all child custom classes.
            b. If no, then go to Step 1.b.ii. You already know the class, but you will need to determine the public methods.
    b. If no, then can you see it with **Show All Classes** checked?
        a. If yes, then you need to adjust the `[ClassList]` section in `extend\JavaEx.ini`.
            a. Expose classes that are ignored by default, which means with **Show All Classes** unchecked, by setting them to `true`.
            b. You may also need to hide some classes that are exposed by default by setting them to `false`. This may be true for tables where you see the individual cells but not the entire table, or for classes that are obscured by container or panel classes.

c. Uncheck **Show All Classes** after you modify the `ClassList`, before recording any classes or window declarations.
d. Go back to Step 1.

b. If no, then that indicates that the component is not derived from AWT Component. You will need to find out from the customer what the custom class is and which public methods are available for that class. The easiest way to determine the public methods is to find it out from the customer, but you can also try using 'javap', which is part of the JDK, to extract the public methods from the class.

a. Can you access the object (ObjectA) of this class (Class A) through a method of a different class (Class B) that SilkTest can recognize? The useful Class B method would only be recorded if you check **Show all methods** when recording the class for the Class B object (ObjectB).

a. If yes, then does the useful Class B method take only 4Test-compatible values as parameters?

a. If yes, then do the methods that you want to call for Class A return 4Test-compatible values and take 4Test-compatible values as parameters?

a. If yes, then you should be able to call *ObjectB*.`invokeMethods`() to call the methods for *ObjectA*: *ObjectB*.`invokeMethods` ({"ClassBMethod", "ClassAMethod"}, {lArgumentsForClassBMethod, lArgumentsForClassAMethod}).
b. If no, then use *ObjectB*.`InvokeJava`(). Within the class that you create for `InvokeJava`(), call the Class B method that returns ObjectA, then call the Class A method.

b. If no, then use *ObjectB*.`InvokeJava`(). Within the class that you create for `InvokeJava`(), call the Class B method that returns ObjectA, then call the Class A method.

b. If no, then use `InvokeJava`(). You will have to find a SilkTest-recognizable object that can indirectly be used to access ObjectA through intermediate objects.

2. Does **Record Class** or the capture class functions give you useful 4Test-accessible methods for the class in question (Class A)?

a. If yes, then call the recorded methods directly in your scripts.
b. If no, then are the methods that you need to use commented out?

a. If yes, then are the methods commented out only because they return values that are not 4Test-compatible, probably because they return custom classes?

a. If yes, then are there methods on the returned classes that return 4Test-compatible values and take only 4Test-compatible values as parameters?

a. If yes, then you should be able to call `invokeMethods`() on the object of interest (*ObjectA*): *ObjectA*.`invokeMethods`({"ClassAMethodOfInterest", "4TestCompatibleMethodForClassReturnedByClassAMethod"}, {lArgumentsForClassAMethod, lArgumentsForMethodForClassReturnedByClassAMethod})
b. If no, then use `InvokeJava`() on the object of interest. Call the non-4Test-compatible methods within the class that you create for `InvokeJava`(). Make sure that you eventually return a 4Test-compatible value from the `InvokeJava`() class.

b. If no, then use `InvokeJava`() on the object of interest. Call the non-4Test-compatible methods within the class that you create for `InvokeJava`(). Make sure that you eventually return a 4Test-compatible value from the `InvokeJava`() class.

c. If no, then use `InvokeJava`() on the object of interest.

How Methods and Properties are Enumerated

This functionality is available only for projects or scripts that use the Classic Agent.

When you record classes, SilkTest enumerates properties and methods as follows:

- By default, SilkTest filters out methods and properties inherited at or above a certain level in a class hierarchy. The threshold at which filtering occurs varies according to the hierarchy. The filtering process makes it easier for you to find the methods and properties you use most frequently. You can turn off this filter to access any of these inherited methods and properties.
- After filtering, SilkTest enumerates only those native methods that accept or return supported Java classes. You can however use unsupported native methods with subclasses that are supported.

Using Native Methods that are not Enumerated

This functionality is available only for projects or scripts that use the Classic Agent.

When you record classes, SilkTest does not enumerate native methods that pass unsupported Java classes as arguments; however, you can use these methods with supported subclasses. Add the prototype by hand in your class definition include file and then use the method in your test script as defined.

Example

If you record a class on a JFC ComboBox, the following native method prototypes are not enumerated because the Java support in SilkTest does not support the `Object` class:

- Object getItemAt(int)
- Object getSelectedItem()
- void setSelectedItem(Object)

If you know that all the items on the JFC ComboBox are instances of a supported class, such as the default Java string class, you can add these prototypes in your class definition include file. Here's how the declarations should look for the Java string class:

- String getItemAt(int)
- String getSelectedItem()
- void setSelectedItem(String)

Thresholds for Filtering Methods and Properties

This functionality is available only for projects or scripts that use the Classic Agent.

In this hierarchy…	Classes are filtered at…
Abstract Windowing Toolkit (AWT)	java.awt.Component and above
Java Foundation Classes (JFC)	com.sun.java.swing.Jcomponent and above

Naming Conflicts

This functionality is available only for projects or scripts that use the Classic Agent.

When you record a class, SilkTest checks for and resolves naming conflicts that arise between 4Test methods and the supported native methods. When naming conflicts arise, make sure you call the appropriate method in your test scripts. The following table shows how SilkTest resolves the naming conflicts:

Type of Naming Conflict	How SilkTest Resolves the Conflict
overloaded native methods	Appends "_n" to overloaded method names to ensure that each is unique, where n is an integer that starts at 2.
native method has the same name as a 4Test method	Prefixes the letter "x" to the name of the native method. For ActiveX/Visual Basic methods only, SilkTest prefixes an underscore character (_) to native methods that begin with "set" and "get" in order to distinguish them from the

Type of Naming Conflict	How SilkTest Resolves the Conflict
	Get and Set methods that SilkTest constructs from properties.
native method uses a reserved 4Test keyword in an inappropriate context	Prefixes the letter "x" to the name of the native method.

When SilkTest prefixes the letter "x" to the name of a native method or property, it also adds the alias keyword with the original name of the native method.

Java Custom Windows

This functionality is available only for projects or scripts that use the Classic Agent.

You do not need to write your own extensions in Java for testing custom Java objects, which are also called `CustomWins`. Instead, the SilkTest Java support lets you access native methods that you can call in your test scripts to manipulate custom Java controls.

You access native methods for a custom Java object by recording a class for that object. To get started, take a look at our guidelines for when and how to record classes.

Loading Class Definition Files or Test Frame Files

This functionality is available only for projects or scripts that use the Classic Agent.

You can use these procedures to load 4Test include files or test frames.

We recommend that you record new class definitions in a new include file or in your test frame file. Do not store these definitions in `javaex.inc`, or another predefined class definition file such as `dotnet.inc`, because we may upgrade *.inc files in future SilkTest releases.

To load class definitions or test frames in selected test scripts

Insert a use statement in each test script that needs to manipulate the controls that you have declared or the objects of the classes that you have defined. Use the following format:

```
use "<directory\file-name>.inc"
```

For example, if your class include file is `custobj.inc` and it resides in the directory `c:\mydir`, insert the following statement:

```
use "c:\mydir\custobj.inc"
```

Recording Classes for Custom Java Controls

This functionality is available only for projects or scripts that use the Classic Agent.

For SilkTest to recognize custom controls, you must record classes for these objects.

The process of recording a custom Java class involves querying the objects, retrieving information on methods and properties for these objects, and then translating this information into 4Test-style prototypes that you can use to write test scripts.

How to Record Classes

Using the Java support in SilkTest, you can use the following approaches to record classes for custom controls in Java applications and applets:

- Record classes for custom Java objects using the recorder.
- Recording classes for custom Java objects from a script.
- Record classes for a window and all of its children using one function call.

Where to store your new class definitions

We recommend that you store your class definitions in a new include file, for example `custobj.inc` or in your test frame file. Do not store these definitions in `javaex.inc`, which is the predefined Java class

definition file, because we will upgrade `javaex.inc` in future versions of our Java support. You will need to load new class include files in SilkTest before testing your application or applet.

If you add, delete, or modify any native methods or parameters for a custom Java class, you need to either re-record the class or modify your class include file to reflect the changes.

Recording Classes for Custom Java Controls Using the Recorder

This functionality is available only for projects or scripts that use the Classic Agent.

Before beginning this procedure, make sure you have taken the necessary prerequisite steps to set up your environment as described in *Configuring SilkTest Java Support* and *Overview of Testing Java Applications and Applets*.

To record new classes for custom controls using the recorder:

1. Start your Java application or applet.
2. Create a new include (.inc) file, open an existing include file, or open the test frame file for storing your new class definitions.
3. Click **Record** > **Class** > **Scripted** to open the **Record Class** dialog box.
4. To include native methods with return or parameter types that do not match valid 4Test methods, check the **Show all methods** check box in the lower left corner of the dialog box.

   You cannot call these methods directly from your 4Test scripts but you can use the `InvokeMethods` or the `InvokeJava` method to call them. When you capture the class, SilkTest displays these methods prefaced by comments in the **Record Class** dialog box. When you click **Paste to Editor**, SilkTest adds the `InvokeMethods` method and these methods, prefaced by comments, to your test script.
5. Position the mouse pointer over the control for which you want to record a class.
6. When the correct name displays in the Window text box, press **Ctrl+Alt**.

   Methods and properties for that class are displayed in the **Record Class** dialog box. If you have checked **Show All Methods**, SilkTest displays these non-4Test methods as comments, that is, prefaced by forward slashes (//).
7. Click the **Derived From** list box to see the list of available 4Test classes. If there is a class type in the list that maps directly to your object, choose it. If not, choose `AnyWin`, which is a generic class.

   For example, if your object is `JavaAwtTextField`, choose `TextField`. If your object is `Spinner`, choose `Anywin`. For additional information, see *winclass Declaration* and derived class.
8. Click **Paste to Editor** to paste the new class into the include file.
9. Uncheck the **Show all methods** check box in the **Record Class** dialog box, if you chose it for this record action.
10. Repeat this procedure for each custom control that does not display in the predefined list of Java classes provided. When you are finished recording classes, click **Close**.
11. Load the include file that contains the new class definitions.

If you find that SilkTest does not recognize some of your custom Java controls, you may need to take additional steps to record classes for these "ignored" objects.

To include native methods with return or parameter types that do not match valid 4Test methods, check the `Show All Methods` check box on the **Record Class** dialog box. SilkTest displays these methods as comments in Methods list.

When you finish recording the class, uncheck the **Show all methods** check box to turn off the recording of all methods. Turning off **Show all methods** when you don't need it helps to keep performance optimal.

Although you cannot call these methods directly from your 4Test script, you can use `InvokeMethods` or `InvokeJavaCode` to call them from your script.

If you add or delete native methods, or modify the parameters of native methods for a custom Java class, you need to either re-record the class or edit your class include file to reflect the changes.

If your test script fails with the error `Function x is not defined for window y`, you might need to modify your window tag from `CustomWin` to the name of your new window class.

Recording Classes for Custom Java Controls from a Script

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest provides two functions that allow you to capture the class information of custom Java controls from a script:

- `CaptureAllClasses`
- `CaptureObjectClass`

The following procedure explains how to use these functions to record new classes for custom Java controls. Before beginning this procedure, make sure you have taken the necessary prerequisite steps to set up your environment as described in *Configuring SilkTest Java Support* and *Overview of Testing Java Applications and Applets*.

To record new classes for custom controls from a script:

1. Start your Java application or applet.
2. Create a new include (.inc) file, open an existing include file, or open an existing test frame file for storing your new class definitions.
3. Open a new script (.t) file.
4. Load the include file `captureclass.inc`, located in the directory where you installed SilkTest.

   Note: We recommend loading this include file only in the scripts that call `CaptureObjectClass` or `CaptureAllClasses`.
5. Open the windows that contain the custom controls and their parent windows.
6. Record declarations for the windows containing your custom controls and paste the declarations into your script file.
7. In your script file, write a `main` routine that calls one or both of the capture functions, according to the following guidelines:

If you want to capture …	Call …
The class for one custom control	`CaptureObjectClass`. For additional information, see the *Script that Calls CaptureObjectClass*.
The class for a custom control and all of its children	`CaptureAllClasses`. For additional information, see the *Script that Calls CaptureAllClasses*.

8. Save and run your script file.
   The results file opens on your desktop and contains the new class definitions.
9. Copy the class definitions from your results file and paste them into the include file you have designated in step 2.
10. Load the include file that contains the new class definitions.

- If you find that SilkTest does not recognize some of your custom Java controls, you may need to take additional steps to record classes for these "ignored" objects.
- If you add or delete native methods, or modify the parameters of native methods for a custom Java class, you need to either re-record the class or edit your class include file to reflect the changes.
- If your test script fails with the error `Function x is not defined for window y`, you might need to modify your window tag from `CustomWin` to the name of your new window class. For the correct sequence of steps to perform before you begin writing test scripts, see *Overview of Testing Java Applications and Applets*.

Recording Classes for Ignored Java Objects

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest ignores certain objects during recording that normally should remain transparent to users, such as panels and containers. Typically these classes don't have a graphical component, or are used solely to aid

the placement of objects. However, there may be cases in which these ignored classes have been extended and contain objects that you want to test. In some situations, custom objects, such as user-defined objects or third-party JavaBeans, might be inadvertently ignored.

Using our Java support in SilkTest, you can expose these ignored objects, then record classes for them in Java applications and in Java applets.

To record classes for ignored Java objects:

1. Start your Java application and make sure Java support is enabled.
2. Create a new include (.inc) file, open an existing include file, or open the test frame file for storing your new class definitions.

   We recommend that you store your new class definitions in a new include file, for example `custobj.inc`, or in your test frame file. Do not store these definitions in `javaex.inc`, the predefined Java class definition file, because we will upgrade `javaex.inc` in future versions of our Java support.
3. Click **Record** > **Class** and then check the **Show all classes** check box in the lower left corner of the dialog box.
4. To include native methods with return or parameter types that do not match valid 4Test methods, check the **Show all methods** check box in the lower left corner of the dialog box.

   You cannot call these methods directly from your 4Test scripts but you can use the `InvokeMethods` or the `InvokeJava` method to call them. When you capture the class, SilkTest displays these methods prefaced by comments in the **Record Class** dialog box. When you click **Paste to Editor**, SilkTest adds the `InvokeMethods` method and these methods, prefaced by comments, to your test script.
5. Position the mouse pointer over the control for which you want to record a class, and when the correct name displays in the Window field, press **Ctrl+Alt**.

   Methods and properties for that class are displayed in the **Record Class** dialog box. If you have checked **Show All Methods**, SilkTest displays these non-4Test methods as comments, that is, prefaced by forward slashes (//).
6. Click the **Derived From** list box to see the list of available 4Test classes.
   - If there is a class type in the list that maps directly to your object, choose it.
   - If not, choose `AnyWin`, which is a generic class. See winclass declaration and derived class for more details.

   Note: Note the Tag for the class you just recorded; you will need this name later.
7. Click **Paste to Editor** to paste the new class into the include file.
8. Uncheck the **Show all classes** check box in the **Record Class** dialog box, and the **Show all methods** check box, if you chose it for this record action, and then click **Close**.

   It is very important to check **Show all classes** only while you are trying to record the class for an ignored Java object.
9. Open `javaex.ini`, located in the extend subdirectory of the directory where you installed SilkTest.
10. In `javaex.ini`, create a section called `[ClassList]` and add a line that reads `<class tag name>=true`.

    For example, if the tag of the class you just recorded is `[com.mycompany.Spinner]`, add this line:

    ```
    com.mycompany.Spinner=true
    ```

    Note: The name can contain wildcards, which can be useful for exposing all classes in a package, for example:

    ```
    com.mycompany.module_classes_to_expose.*\=true
    ```
11. Save and close `javaex.ini`.

    When you uninstall SilkTest, the file `javaex.ini` is backed up as `javaex.bak` in the `<SilkTest install directory>\extend` folder. Any changes you made to `javaex.ini` can be reinstated by

copying them from `javaex.bak` and pasting them into the new `javaex.ini` that is created when you reinstall SilkTest.

12. Restart the Agent and your application.
13. Load the include file that contains the new class definitions.

Recording Java Window Declarations

This functionality is available only for projects or scripts that use the Classic Agent.

To record Java window declarations:

1. Create a new test frame file.

   SilkTest loads a new test frame file by automatically specifying its full path in the **Use Files** text box of the **Runtime Options** dialog box. Instead of creating a new test frame, you can open a test frame file that you already created for this Java test script or suite. If you use an existing test frame file that has not been loaded, load the test frame file now.
2. If you recorded classes for any Java controls, load the class definition file now.
3. With your test frame file as the active window, choose **Record** > **Window Declarations** and record declarations.

You can now use multitags when recording window declarations for Java applications. However, additional considerations must be made for top-level windows.

Turning On the Class Declaration Filter

This functionality is available only for projects or scripts that use the Classic Agent.

When you record classes, SilkTest by default filters out properties and methods inherited at or above a certain level in a class hierarchy. The threshold at which filtering occurs varies according to the hierarchy.

You can turn off this filter if you want these properties and methods to be enumerated. If you then want to turn the filter back on, follow the procedure described below.

Note: We provide this filter as a convenience, but it has not been thoroughly tested.

To turn on the class declaration filter:

1. Open the `javaex.ini` file.

   The path is `<SilkTest install directory>\extend\javaex.ini`.
2. Either remove the declaration `FilterClassDecl=false` from the `[Recording]` section, or change the declaration to `FilterClassDecl=true`.
3. Restart the Agent, and the application or applet under test.

Turning Off the Class Declaration Filter

This functionality is available only for projects or scripts that use the Classic Agent.

When you record classes, SilkTest by default filters out properties and methods inherited at or above a certain level in a class hierarchy. The threshold at which filtering occurs varies according to the hierarchy. You can turn off this filter if you want these properties and methods to be enumerated.

We provide this filter as a convenience, but it has not been thoroughly tested.

To turn off the class declaration filter:

1. Open the `javaex.ini` file.

   The path is `<SilkTest install directory>\extend\javaex.ini`.
2. In the `[Recording]` section, add the command `FilterClassDecl=false`.
3. Save and close `javaex.ini`.
4. Restart the Agent, and the application or applet under test.

Extending Java Support

Keeping the DOS Window Open when Returning to DefaultBaseState

This functionality is available only for projects or scripts that use the Classic Agent.

Java applications running in Windows are launched from DOS.

You do not need to perform this task if appstate is set to "none".

To keep the DOS window open when returning to base state:

1. Launch your Java application.

   If you are running JDK, your DOS window is minimized; if you are running JRE, your DOS window is not minimized.
2. Click **RecordWindow Identifiers** in the SilkTest menu bar.
3. Click the DOS window. If the DOS window is minimized, restore it first. Place your cursor over the title bar of the DOS window and press **Ctrl+Alt** to record the identifier.
4. Open your test frame file and expand the main window declaration for the Java application you are testing.
5. Un-comment the line that begins `const lwLeaveOpen` and select the text ? near the end of that line.
6. In the **Record Windows Identifiers** dialog box, click **Paste to Editor** to insert the DOS window identifier as the value for `lwLeaveOpen`, inside the {} brackets.
7. Close the **Record Windows Identifiers** dialog box and save the test frame file.

Redirect Output from Java Console to File

This functionality is available only for projects or scripts that use the Classic Agent.

When you enable Java support, you can redirect Java console output to a local file, where you can more easily scroll and copy the text.

To redirect Java console output to a local file:

1. Choose **Options** > **Extensions** from the SilkTest menu bar.
   The **Extensions** dialog box opens.
2. Select and highlight an enabled Java application or browser.
3. Click **Java**.
   The **Extension Options** dialog box opens.
4. Check the **Redirect Java Console Output** check box.
5. In the **Java Console File Name** text box, enter the path to the file where you want to redirect Java console output.
6. Click **OK**.

Note: SilkTest hangs if you are using Java Plug-In for JVM 1.4.2 and call `Browser.Close()` with the Java Console open.

Using Java Database Connectivity (JDBC)

This functionality is available only for projects or scripts that use the Classic Agent.

To verify information from an SQL-compliant database, you might want to hook into JDBC to access the data. You can access JDBC directly by using the `InvokeJava` method to call a Java class that makes a series of JDBC calls and returns the data of interest.

Invoking Java Applications and Applets

Invoking Java Applets

This functionality is available only for projects or scripts that use the Classic Agent.

After you configure SilkTest for Java support and enable the Java extension, you can invoke the Java applet from within a supported browser. For additional information, see *Enabling Extensions*.

Invoking JDK Applications

This functionality is available only for projects or scripts that use the Classic Agent.

Once you configure SilkTest for testing standalone Java applications, you can invoke JDK applications as you normally would from the command line.

To invoke JDK applications using -classpath

Enter the following command:

```
java -classpath <Java support path>;<other directories, if any> <name of application>
```

Example

Assuming you installed SilkTest in the default directory `c:\Program Files\Silk\SilkTest`, that you are using JDK 1.3 as your Java Virtual Machine (JVM), and your CLASSPATH contains only the Java support path, you would launch the application MyJDKapp by entering:

```
java -classpath c:\Program Files\Silk\SilkTest\JavaEx\SilkTest_Java3.jar MyJDKapp
```

To invoke JDK applications without using command line arguments

Enter the following command:

```
java <name of application>
```

Example

To invoke the application MyJDKapp, enter:

```
java MyJDKapp
```

Invoking JRE Applications

This functionality is available only for projects or scripts that use the Classic Agent.

Once you set CLASSPATH for testing standalone Java applications, you are ready to invoke your application using the Java Runtime Environment (JRE).

Note: The JRE ignores the CLASSPATH environment variable. As a result, you must invoke JRE applications with command line arguments to pick up the value of CLASSPATH.

The following table describes the commands you can use:

Command	Description
-cp	Searches first through directories and files specified, then through standard JRE directories.
-classpath	Ignores the value of your CLASSPATH environment variable. You must specify a complete search path on the command line. Does not search the standard JRE directories.

To invoke JRE applications using -cp

Enter the following command:

```
jre -cp %CLASSPATH%;<other directories, if any> <name of application>
```

Example

Assuming your CLASSPATH is set to the complete search path including the Java support path, you would launch the application MyJREapp by entering:

```
jre -cp %CLASSPATH% MyJREapp
```

Invoking JRE Applications using -classpath

This functionality is available only for projects or scripts that use the Classic Agent.

To invoke JRE applications using –classpath, enter the following command:

```
jre -classpath <Java support path>;<other directories, if any>
```

Example

Assuming you installed SilkTest in the default directory `c:\Program Files\Silk\SilkTest`, you are using JRE 1.1.5 as your Java Virtual Machine (JVM), and your CLASSPATH contains only the Java support path, you would launch the application MyJREapp by entering:

```
Java -classpath c:\Progra~1\Silk\SilkTest\JavaEx
\SilkTest_Java3.jar MyJREapp
```

Invoking Symantec Visual Café Applications from Command Line

This functionality is available only for projects or scripts that use the Classic Agent.

When you invoke Visual Café applications from the command line, the *CLASSPATH* environment variable is ignored. Therefore, to tell Visual Café about the Java support path, you must use the -classpath argument on the command line to specify the locations of the following class libraries:

Library	Path
Java support	Java support path
AWT and java.* classes	`<Visual Café install directory>\java\lib\classes.zip`
Symantec Itools classes This library is required only if your application uses Itools	`<Visual Café install directory>\bin\components\SymBeans.jar`

To invoke Visual Café applications from the command line:

1. Move to the directory where your Visual Café executable resides or put this directory on your path. The directory is `<Visual Café install directory>\java\bin\`.
2. Enter the following command:

```
java.exe -classpath <Java support path>;
  <Visual Café install directory>\java\lib\classes.zip;
  <Visual Café install directory>\bin\components\SymBeans.jar <application>
```

Sample Visual Café Command Line

This functionality is available only for projects or scripts that use the Classic Agent.

Assuming the following conditions:

- You install SilkTest in `c:\Silk`
- You install Visual Café 2.0+ in `c:\Symantec`
- Your application uses Itools controls
- Your application is MyApp.class

Then, your Visual Café command line should look like the following:

Invoke Symantec Visual Café Applications from IDE

This functionality is available only for projects or scripts that use the Classic Agent.

Before you invoke Visual Café applications from the Integrated Development Environment (IDE), you must first add the Java support path to the CLASSPATH by editing the `sc.ini` file.

To invoke Visual Café applications from the IDE:

1. Make sure SilkTest is configured to test Java.
2. Open the `sc.ini` file in your favorite text editor.
   The file is located in `<Visual Café install directory>\bin\sc.ini`.
3. On the line that begins with `CLASSPATH=`, add the Java support path.
   For example if you installed SilkTest in `c:\Silk` and you are using Visual Café 2.0 as your Java Virtual Machine (JVM), add the following Java support path to the end of the CLASSPATH:
   ```
   ;c:\Silk\SilkTest_Java3.jar
   ```
   For additional information, refer to the VisualCafe documentation.
4. Save and close `sc.ini`.
5. Restart Visual Café.
6. Click **Project** > **Execute** to invoke your application within the IDE.

Accessing Java Objects and Methods

Accessing Native Methods for Predefined Java Classes

This functionality is available only for projects or scripts that use the Classic Agent.

In some situations, you might want to access the native methods for predefined Java classes, for example if a particular function was not supported in 4Test, but could be performed using a native method. You can access the native methods for controls that are part of predefined Java classes by re-recording the class for the control.

To access native methods for predefined Java classes:

1. Start your Java application or applet, and SilkTest.

2. Open the predefined class definition file, javaex.inc., and comment out the predefined definitions for classes whose native methods you want to access.
3. Create a new include (.inc) file or open an existing include file for storing your new class definitions.
4. Click **Record** > **Class** > **Scripted** to open the **Record Class** dialog box.
5. Position the mouse pointer over the predefined control for which you want to record a new class.
6. When the correct name displays in the Window field, press Ctrl+Alt. Methods and properties for that class are displayed in the Record Class dialog.
7. Click the **Derived From** list box to see the list of available 4Test classes. If there is a class type in the list that maps directly to your object, choose it. If not, choose **AnyWin**, which is a generic class. See winclass declaration and derived class for more details.
8. Click **Paste to Editor** to paste the new class into the include file.
9. Repeat this procedure for each predefined class whose native methods you want to access. When you are finished recording classes, click **Close**.
10. Load the class include file that stores your new class definitions.

Accessing JavaScript Methods in Netscape Navigator

This functionality is available only for projects or scripts that use the Classic Agent.

Generally, SilkTest does not recognize methods defined for JavaScript objects on a Web page if you are testing an application that uses the Classic Agent. However, Netscape allows you to access JavaScript objects on a Web page, but only if the page contains an applet. You can leverage this capability by using the `InvokeJava` method to first access an applet running in Netscape and then move down the hierarchy to the JavaScript object and its methods.

Accessing Nested Java Objects

This functionality is available only for projects or scripts that use the Classic Agent.

Sometimes you cannot retrieve 4Test-compatible information about a Java control with a single call to a 4Test method; instead, you need to call several nested methods, each returning an intermediate object to be passed to the next method. If any of these methods returns intermediate results that are not 4Test-compatible, you will not be able to perform these nested calls from 4Test.

In such cases, you can use two alternative methods:

Method	What it does	How to use it
InvokeJava	Allows you to invoke a Java class from 4Test for manipulating a nested Java object.	See this code example: *Retrieve Item Text in JFC JTree Object (using InvokeJava)*
invokeMethods	Allows you to perform nested calls inside Java, even if intermediate results are not 4Test-compatible	See this code example: *Draw a Line in a Multiline Text Field*

Accessing Non-Visible Java Objects

This functionality is available only for projects or scripts that use the Classic Agent.

Currently, SilkTest cannot enumerate or manipulate Java objects that are not derived from the Abstract Windowing Toolkit (AWT) Component object.

With the `InvokeJava` method, you can access these non-visible objects by performing the following steps:

1. A development group adds a Java class to an application that provides methods for accessing object references to non-visible objects of interest.
2. Create the Java class to be invoked by the `InvokeJava` method.
3. In the Java class that is invoked by `InvokeJava`, call the access methods that provide a reference to the non-visible objects of interest in your application.

4. Manipulate the non-visible objects as desired.

Calling Nested Methods

This functionality is available only for projects or scripts that use the Classic Agent.

Sometimes you cannot retrieve 4Test-compatible information about a Java control with a single call to a 4Test method; instead, you need to call several nested methods, each returning an intermediate object to be passed to the next method. If any of these methods returns intermediate results that are not 4Test-compatible, you will not be able to perform these nested calls from 4Test.

In such cases, we provide two ways for you to call nested methods:

- Use `InvokeJava`, a method that allows you to invoke Java classes from 4Test for manipulating nested Java objects.
- Use `invokeMethods`, a method that allows you to perform nested calls inside Java, even if intermediate results are not 4Test-compatible. You can call `invokeMethods` for any Java object as long as you add the `invokeMethods` prototype inside the object's class declaration.

Identifying Custom Controls

This functionality is available only for projects or scripts that use the Classic Agent.

To identify custom Java controls:

1. Click **Record** > **Window Declarations**.
2. Pass your cursor over the controls you want to test. You will be able to identify the custom controls by their class, which appears as CustomWin in the **Record Windows Declarations** dialog box. You can also press **Ctrl+Alt** to pause tracking and view the controls you want to test.

Ignoring a Java object

This functionality is available only for projects or scripts that use the Classic Agent.

If you are using the Java extension and you want to ignore a Java class, object, or container, you must edit your `javaex.ini` file:

1. Open `javaex.ini`, located in the `<SilkTest install directory>\extend` folder.
2. Add `myclass=false` as a new line to the file, where myclass is the full class name.

   For example, if the tag of the class you just recorded is `[com.mycompany.Spinner]`, add the following line:

   ```
   com.mycompany.Spinner=false
   ```

   Note: The name can contain wildcards, which can be useful for ignoring all classes in a package, for example `com.mycompany.module_classes_to_ignore.*\=false`. You must use the value `false` and not the value `ignore`.

You might find that ignoring a top level object causes all objects underneath it to be ignored as well. If this is an issue, you can mark the top level object as `false` and add any objects that you do not want to be ignored to the ClassList and set to the value to true. For example:

```
com.ignore.this.object = false
 com.dontignore.this.object = true
```

The second object displays in the GUI as a child of the first.

Testing Java Scroll Panes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

A scroll pane in Java is a container that holds a single child component. If the scroll pane is smaller than the child component, you can scroll vertically and horizontally to see all parts of that component.

The state of the scroll bars in a scroll pane is managed by internal objects that implement the `Adjustable` interface. To manipulate the scroll bars, you must first get an `Adjustable` object, and then use `Adjustable` and scroll bar methods to move them.

To test scroll bars in a scroll pane, use `invokeMethods`, a method that allows you to perform nested calls inside Java to access `Adjustable` objects.

Frequently Asked Questions

Frequently Asked Questions about the Java Extension

This functionality is available only for projects or scripts that use the Classic Agent.

We have responded to frequently asked questions about our Java support, including. This information answers questions that might arise as you use and become familiar with Java support. We have provided answers to the following questions about classpath and testing applications and applets:

- Why do I need to disable the classpath if I have Java installed, but am not testing it?
- Can I invoke Java code from 4Test scripts?
- Can I use the Java plug-in to test applets outside my browser's native JVM?
- Can I test JavaScript objects?
- Why do I see so many Java CustomWin objects?
- When should I record classes?
- How do I decide whether to use 4Test methods or native methods?
- How can I save the changes I make to `javaex.ini`?
- Can I record AWT popup menus?

Why do I See so many Java CustomWin Objects

This functionality is available only for projects or scripts that use the Classic Agent.

Objects that do not belong to any of our predefined Java classes are custom controls, which are identified as `CustomWin` objects by SilkTest. Most Java applications and applets use many custom controls to fine tune functionality and the user interface.

To manipulate a custom Java object for testing, you do not need to write your own extensions. Instead, you can use the object's own native methods and properties. Our Java support lets you access native methods and properties, by recording classes for custom controls. To get started, see the *Guidelines for When and How to Record Classes*.

Why do I Need to Disable the Classpath if I have Java Installed but am not Testing it

This functionality is available only for projects or scripts that use the Classic Agent.

If you are not testing Java but do have Java installed, we recommend that you disable the classpath before using SilkTest. If you do not disable the classpath, SilkTest still checks for a Java classpath every time you run a test plan. To do this at the time of install, select **None** on the **Java** dialog box. To make sure that you have disabled the classpath after installation, verify that the path to the Java extension is disabled in the Java variable, which is stored in the system variables.

For example, to do this on Windows 7, click **Start** > **Control Panel** > **System and Security** > **System**. Then click **Properties**. Click **Advanced System Settings** to open the **System Properties** dialog box. Finally, click **Environment Variables** to open the **Environment Variables** dialog box. In the **System variables** area, select the Java variable and disable the path to the Java extension, by placing an underscore at the beginning of the path.

When should I Record Classes

This functionality is available only for projects or scripts that use the Classic Agent.

Generally, you should record classes when you need to manipulate an object in the Java application or applet under test that is seen as a `CustomWin` by SilkTest. For additional information, see the *Guidelines for When and How to Record Classes*.

How do I Decide whether to Use 4Test Methods or Native Methods

This functionality is available only for projects or scripts that use the Classic Agent.

When both 4Test methods and native methods are available for all controls you want to test, we recommend using 4Test methods in your test scripts. We do not recommend mixing 4Test and native methods because of incompatibilities between Java and 4Test.

For more information, review our recommendations for when to use 4Test versus native Java controls.

How can I Save the Changes I make to javaex.ini

This functionality is available only for projects or scripts that use the Classic Agent.

When you uninstall SilkTest, the file `javaex.ini` is backed up as `javaex.bak` in the extend subdirectory of your install location. Any changes you made to `javaex.ini` can be reinstated by copying them from `javaex.bak` and pasting them into the new `javaex.ini` that is created when you reinstall SilkTest.

How can I Record AWT Menus

This functionality is available only for projects or scripts that use the Classic Agent.

You cannot use the `JavaAwtPopupMenu` class to record AWT menus. It is available for playback only. You must manually script any interaction with AWT menus.

Can I Use the Java Plug-In to Test Applets Outside my Browsers Native JVM

This functionality is available only for projects or scripts that use the Classic Agent.

For testing purposes, you can use the Java plug-in to run applets outside the native Java virtual machine of your browser.

Can I Test JavaScript Objects

With the Classic Agent, you can use `InvokeJava` to access methods for testing JavaScript objects in Netscape, if these objects reside on a Web page that contains an applet.

With the Open Agent, you can use `ExecuteJavaScript` to test anything that uses JavaScript.

Can I Invoke Java Code from 4Test Scripts

This functionality is available only for projects or scripts that use the Classic Agent.

With our Java support, you can invoke Java code from 4Test scripts using the method `InvokeJava`.

Testing Java SWT and Eclipse Applications

Overview of Java SWT Applications and Eclipse Support

This functionality is available only for projects or scripts that use the Open Agent.

SilkTest provides built-in support using the Basic Workflow for testing applications that use widgets from the Standard Widget Toolkit (SWT) controls. When you configure a Java SWT/RCP application, SilkTest automatically provides support for testing standard Java SWT/RCP controls.

SilkTest supports:

- Testing Java SWT controls embedded in Java AWT/Swing applications as well as Java AWT/Swing controls embedded in Java SWT applications.
- Standalone SWT applications that use the EXT or CLASSPATH configuration.
- Testing Java SWT applications that use the IBM JDK or the Sun JDK.
- Any Eclipse-based application that uses SWT widgets for rendering. SilkTest supports both Eclipse IDE-based applications and RCP-based applications.

Sample Applications

SilkTest provides sample SWT test applications. You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Java SWT** > **SWT Test Application <version>**. Select the sample application that is right for your environment.

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

For instructions on running the sample applications, see Running the sample SWT applications. You can also access Java tutorials by clicking SilkTest Tutorials.

Object Recognition

Java SWT applications support dynamic object recognition.

When you record a testcase with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Existing testcases that use dynamic object recognition without locator keywords in an INC file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object recognition without locator keywords in an INC file.

Using custom class attributes becomes even more powerful when it is used in combination with dynamic object recognition.

Supported Controls

For a complete list of the widgets available for SWT testing, see Supported SWT Widgets for the Open Agent.

For a complete list of the record and replay controls available for Java SWT testing, view the `SWT.inc` and `JavaSWT.inc` file. To access the `JavaSWT.inc` file that is used with the Open Agent, navigate to the `<SilkTest directory>\extend\JavaSWT` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\JavaSWT\JavaSWT.inc`. To access the `SWT.inc` file, navigate to the `<SilkTest directory>\extend\` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\SWT.inc`.

Running the Sample SWT Applications

SilkTest provides two sets of sample applications for SWT:

- Those designed to work with the Open Agent.
- These sample applications do not require any manual configuration. During setup, SilkTest determines if there is a JRE installed on the system and enables the samples to use that JRE. If no JRE is found, SilkTest enables the samples to use the JRE that SilkTest provides. You can use any JRE that you require with these sample applications.
- Those designed to work with the Classic Agent.
- These sample applications were provided with earlier versions of SilkTest and may require manual configuration. You can use any JRE that you require with these sample applications.

If you want to use the sample applications designed for the Open Agent with the Classic Agent, you must use a JRE other than the one provided by SilkTest. To do this, you may need to update the Java reference in the *.bat file that launches each sample application. The sample applications designed for the Classic Agent can be used with the Open Agent. Keep in mind that the controls in the sample applications differ because each Agent supports different controls.

To run the sample SWT applications

1. Download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*.

2. After you have installed the sample applications, choose **Programs** > **<SilkTest program group>** > **Sample Applications** > **Java SWT**.
3. To use the SWT Test Application with the Open Agent, choose SWT Test Application 3.2, SWT Test Application 3.3, or SWT Test Application 3.4.
4. To use the SWT Test Application with the Classic Agent, choose Classic Agent and then choose the sample application.

Updating the Java SWT Batch File for the Sample Application

SilkTest provides several sample SWT applications. In most cases, the sample applications designed to work with the SilkTest Open Agent should not require any manual configuration.

You may need to edit the configuration file:

- To change the JRE that SilkTest uses.
- For the sample applications designed to work with the Classic Agent.
- To run the sample applications designed for the Open Agent with the Classic Agent.

To update `SWTTestApp.bat`:

1. Use Windows Explorer to navigate to `\SilkTest\JavaEx\SWTTest\SWTTestApp.bat`.

   By default this directory is located at `C:\Program Files\Silk\SilkTest\JavaEx\SWTTest \SWTTestApp.bat`.
2. Open `SWTTestApp.bat` with a text editor (such as Notepad).
3. Add the following line to the beginning of the .bat file (it must be the very first line): `set JavaRun="c: \Program Files\Java\SWT_JRE\bin\java.exe"`
4. Edit the line from `java.exe -classpath SWTTestApp.jar qapswttest.TestApp` to `%JavaRun % -classpath SWTTestApp.jar qapswttest.TestApp`.
5. Save and close the `SwtTestApp.bat` file.
6. Run the sample SWT application.

Suppressing Controls for Certain Classes

You can suppress the controls for certain classes for .NET, Java SWT, and Windows API-based applications. For example, you might want to ignore container classes to streamline your testcases. Ignoring these unnecessary classes simplifies the object hierarchy and shortens the length of the lines of code in your test scripts and functions. Container classes or ‘frames’ are common in GUI development, but may not be necessary for testing.

Common classes that are suppressed include:

.NET	Group
Java SWT	org.eclipse.swt.widgets.Composite
	org.eclipse.swt.widgets.Group
Windows API-based applications	Group

To suppress the controls for .NET applications, you must use the Classic Agent. To suppress the controls for Java SWT and Windows API-based applications, you must use the Open Agent.

1. Click **Options** > **Class Map**.
   The **Class Map** dialog opens.
2. In the Custom class field, type the name of the class that you want suppress.

The class name depends on the technology and the extension that you are using. For Windows API-based applications, use the Windows API-based class names. For Java SWT applications, use the fully qualified Java class name.

To ignore the **SWT_Group** in a Windows API-based application, type `SWT_Group`. For Java SWT applications, type `org.eclipse.swt.widgets.Group`.

3. In the **Standard class** list, select **Ignore**.
4. Click **Add**. The custom class and the standard class display at the top of the dialog.

Using the Open Agent

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.
7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.

8. Click Finish.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

    SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

    SilkTest opens the include file.
11. Record the test case whenever you are ready.

Custom Attributes

Add custom attributes to a test application to make a test more stable. You can use custom attributes with the following technologies:

- Java SWT
- Swing
- WPF
- xBrowser
- Windows Forms
- Win32
- SAP

For example, in Java SWT, the developer implementing the GUI can define an attribute (for example, `silkTestAutomationId`) for a widget that uniquely identifies the widget in the application. A tester using SilkTest can then add that attribute to the list of custom attributes (in this case, `silkTestAutomationId`), and can identify controls by that unique ID. Using a custom attribute is more reliable than other attributes like caption or index, since a caption will change when you translate the application into another language, and the index will change whenever another widget is added before the one you have defined already.

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

To create tests that use custom class attributes, you must use the Open Agent. For the following technologies, you can include custom attributes in tests that use hierarchical or dynamic object recognition:

- Java SWT
- Swing
- WPF
- Windows Forms
- Win32
- SAP

Because xBrowser only supports tests that use dynamic object recognition, you can only create tests that use dynamic object recognition. First, enable custom attributes for your application and then create the test.

Recording tests that use dynamic object recognition

Using custom class attributes becomes even more powerful when it is used in combination with dynamic object recognition. For example, If you create a button in the application that you want to test using the following code:

```
Button myButton = Button(parent, SWT.NONE);
myButton.setData("SilkTestAutomationId", "myButtonId");
```

To add the attribute to your XPath query string in your testcase, you can use the following query:

```
Window button = Desktop.Find(".//
PushButton[@SilkTestAutomationId='myButton']")
```

Recording tests that use hierarchical object recognition

This method is supported for Java SWT only.

When you record a test that contains a custom attribute, SilkTest records the custom attribute and includes the information in the frame.inc file. For example, the Java SWT sample application contains a custom attribute, `silkTestAutomationId` that is defined for the top-level window. When you record a testcase of the main window using `silkTestAutomationId` as the class attribute, the frame file contains the following:

```
window Shell SwtTestApplication
"&SilkTestAutomationId=shSWTTestApp"
```

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.

The attributes that SilkTest supports for Java SWT include:

- caption (supports wildcards ? and *)
- all custom object definition attributes

Attribute names are case sensitive.

Dynamically Invoking Java Methods

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

Call dynamic methods on objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a control, use the `GetDynamicMethodList`() method.

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

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty`() method. To retrieve a list of supported dynamic properties for a control, use the `GetPropertyList`() method.

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

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that SilkTest supports for the control.
- All public methods of the SWT, AWT, or Swing widget.

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

Supported Parameter Types

The following parameter types are supported:

All built-in SilkTest Classic types.	SilkTest Classic types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
Enum types.	Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the enum type, java.sql.ClientInfoStatus you can use the string values of REASON_UNKNOWN, REASON_UNKNOWN_PROPERTY, REASON_VALUE_INVALID, or REASON_VALUE_TRUNCATED.
Other controls.	Control parameters can be passed as WINDOW.

Example

A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

```
customControl.Invoke("Reset")
REAL sum = customControl.DynamicInvoke("Add",{1,2})
```

Testing with the SilkBean Applications

Overview of the SilkBean

This functionality is available only for projects or scripts that use the Classic Agent.

Using the SilkBean, you can test standalone Java applications on non-Windows platforms, such as UNIX and Linux. You can perform cross-platform testing of 100% pure Java controls in standalone Java applications in a number of test environments. SilkBean provides flexibility that enables you to:

- Test a single standalone Java application on a non-Windows target machine.
- Set up multiple testing sessions on the same Windows host machine to test multiple standalone Java applications on the same non-Windows target machine.
- Set up multiple testing sessions on different Windows hosts to test multiple standalone Java applications on the same target machine.

When using SilkBean, you create all functions using SilkTest on your Windows host machine, and then play back the scripts on a non-Windows target machine running SilkBean.

SilkBean runs on the following certified platforms:

- Solaris 2.5 or later.
- Redhat Linux 6.0 and 6.2.
- Hewlett-Packard UNIX (HP-UX) 10.2 and 11.0.
- Advanced Interactive Executive (AIX) 4.3.2 and 4.3.3.

For details about supported versions, refer to the *Release Notes* in **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

Configuring SilkBean

Preparing Test Scripts for Running with the SilkBean

This functionality is available only for projects or scripts that use the Classic Agent.

When you are preparing test scripts to run with the SilkBean, keep the following tips in mind:

- Do not use ~ActiveApp in window declarations that will be used with SilkBean. Either use a generic parent tag, such as `[JavaMainWin]#1/` in the tag statement, or use a tag function if the parents can occur at different levels in the window hierarchy. For details see the multitag statement topic.
- **Note:** In certain cases where dialog boxes are parented to other dialog boxes, the window hierarchy may differ between SilkBean and Windows. As for ~ActiveApp, use a tag function to compensate for the different levels of the parent windows.
- Do not call `Desktop.GetActive()` in your scripts, since it is invalid for SilkBean.
- The index tag of a MoveableWin, for example a main window or dialog box, may differ between SilkBean and the standard Java extension. The difference is not due to a difference in operating systems, like UNIX and Windows, but rather to a difference between the Java extension and SilkBean. For example, the Java extension may see the top visible window as "#1", but SilkBean may see the bottom-most window, which was the first one created, as "#1".
- If you must use an index tag, then you can use a conditional expression in the tag to accommodate both the Java extension and SilkBean. The condition should be based on the value of the OPT_USE_SILKBEAN Agent option, which is TRUE if SilkBean is currently being used. For example, if the index for the Java extension is #1 and the index for SilkBean is #3, use:

```
tag "#{Agent.GetOption (OPT_USE_SILKBEAN) ? "3" : "1"}"
```

- Insert a right-mouse Click() before JavaAwtPopupMenu or JavaJFCPopupMenu in order to bring up the menu.

Additional considerations when testing AWT

- If your application contains Abstract Windowing Toolkit (AWT) menus or AWT menu items, modify your declarations for these controls in a new test frame file as follows:

If the AWT declaration is …	Change the declaration to …
Menu	JavaAwtMenu
MenuItem	JavaAwtMenuItem

- If you are testing AWT controls, do not use low-level methods to simulate mouse and keyboard events. Due to the limitations of platform-specific implementations of AWT controls, low-level events are not supported. Instead, use high-level methods when possible.

Java Foundation Class (JFC) controls support low-level events.

Configuring SilkBean Support on the Target (UNIX) Machine

This functionality is available only for projects or scripts that use the Classic Agent.

This section contains instructions for configuring the target UNIX machine when the test application is running in Java 2 environments, which means JDK/JRE versions greater than or equal to 1.2.

1. Make sure that JDK 1.2 or later is installed on the UNIX machine and that the path to its "bin" directory is included in the PATH variable.
2. Copy the following files from your SilkTest installation on the Windows machine onto the UNIX machine:
   - Copy `SilkTest_Java3.jar` to the JVM's `lib/ext` directory.
   - Copy `access3bean.prop` to the JVM's lib directory and rename it to `accessibility.properties`.
3. Start the SilkBean using the following command:

```
java segue.server.SilkBean debug <port number> &
```

   - Include the optional `debug` parameter if you want to run the SilkBean server in debug mode.

- The port number defaults to 2966, if it is not specified.
- The ampersand (&) at the end of the line should only be used on an UNIX target machine. It specifies that the SilkBean should run in the background.

4. Start the test application manually or from the script.

Note: There are two SilkBean-specific options for the Java command line that is used to start the AUT. The options are specified using the '-D' switch.

ST_CONN_TIMEOUT	The maximum time (in seconds) allowed for connection between the SilkBean and the application. If unspecified, the default value of 30 seconds is used.
qap.port	The port through which to connect SilkBean. The default is 2966. This number must match the port number specified in the SilkBean command line. `java segue.server.SilkBean <port number> &`.

Example

For example, to start the Java application `myapp.jar` with port number 2970 and a connection timeout of 60 seconds, use:

```
java -Dqap.port=2970 -DST_CONN_TIMEOUT=60 myapp.jar
```

Configuring SilkBean Support on the Host Machine when Testing Multiple Applications

This functionality is available only for projects or scripts that use the Classic Agent.

If you are testing multiple applications on the same machine, or different platforms, you must make the following changes on the host machine to enable the Agent to interact with the SilkBean running on the UNIX machine:

Enable SilkBean by adding the following code to the script file:

```
Agent.SetOption(OPT_USE_SILKBEAN, FALSE)
Agent.SetOption(OPT_SET_TARGET_MACHINE, "targetmachine:port#"))
Agent.SetOption(OPT_USE_SILKBEAN, TRUE)
```

Troubleshooting SilkBean

Correcting Problems when Using the SilkBean

This functionality is available only for projects or scripts that use the Classic Agent.

General help

For general help when testing Java applications with the SilkBean, you can start up the SilkBean in debug mode on the target machine. Then, look for AppRegistered debug messages to display after invoking your Java application.

Specific workarounds

There are workarounds for the following specific problems:

- I cannot test the AWT FileOpen dialog box.
- I cannot pick AWT menus.
- `SetActive`, `Minimize`, and `Restore` methods do not work with SilkBean.
- I cannot select menu items in JFC cascaded menus.
- SilkBean cannot find a main window or a dialog box declared with an index tag.
- I cannot redirect console output to a text file.

Testing .NET Applications

SilkTest Classic provides built-in support for testing .NET applications.

Built-in support for Windows Forms is provided only for projects or scripts that use the Open Agent.

For information about the supported versions, refer to the *SilkTest Release Notes*.

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.

The attributes that SilkTest supports for Windows Forms include:

- automationId
- caption (supports wildcards ? and *)
- 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.)

Attribute names are case sensitive.

Windows Forms Applications

Overview of Windows Forms Applications

SilkTest provides built-in support for testing .NET Windows Forms (Win Forms) applications using the Open Agent. SilkTest can record and play back controls embedded in:

- Framework version 2.0
- Framework version 3.0
- Framework version 3.5
- Framework version 4.0

SilkTest also provides built-in support for testing .NET standalone and No-Touch Windows Forms (Win Forms) applications using the Classic Agent. However, side-by-side execution is supported only on standalone applications.

Sample Applications

SilkTest provides sample standalone Windows Forms applications. You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Microsoft .NET** to select the sample application that is right for your environment.

SilkTest contains the following sample .NET test application:

- Windows Forms Test Application - runs against the 2.0 Framework or later versions.

By default the test application run in standalone mode, but you can also use them as a No-Touch sample application.

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

Object Recognition

Win Forms applications support dynamic object recognition. This functionality is available only for projects or scripts that use the Open Agent.

When you record a test case with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Existing test cases that use dynamic object recognition without locator keywords in an INC file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object recognition without locator keywords in an INC file.

The name that was given to an element in the application is used as `automationId` attribute for the locator if available. As a result, most objects can be uniquely identified using only this attribute.

Supported Controls

For a complete list of the record and replay controls available for Win Forms testing with the Open Agent, see the *Windows Forms Class Reference*.

SilkTest does not support Win Forms controls embedded in other applications, for example, Internet Explorer.

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.

The attributes that SilkTest supports for Windows Forms include:

- automationId
- caption (supports wildcards ? and *)
- 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.)

Attribute names are case sensitive.

Dynamically Invoking Windows Forms Methods

This functionality is available only for projects or scripts that use the Open Agent.

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

Call dynamic methods on objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a control, use the `GetDynamicMethodList`() method.

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

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty`() method. To retrieve a list of supported dynamic properties for a control, use the `GetPropertyList`() method.

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

- All built-in SilkTest types.

  SilkTest types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
- Enum types.
- Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visibility` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
- .NET structs and objects.

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

  Control parameters can be passed as WINDOW.

Returned Values

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

- The correct value for all built-in SilkTest types.
- All methods that have no return value return NULL.

Example

A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

```
customControl.Invoke("Reset")
REAL sum = customControl.DynamicInvoke("Add",{1,2})
```

The calculator control also has a `LastCalculationResult` property. You can use the following code to read the property:

```
REAL lastResult =
customControl.GetProperty("LastCalculationResult")
```

Prerequisites for Testing .NET No-Touch Applications

This functionality is available only for projects or scripts that use the Classic Agent.

Before you can use SilkTest to test .NET no-touch applications, you must:

- Have System Administrator privileges on the SilkTest Agent's machine.
- Have either the 2.0 or 1.1.NET Framework installed; these Frameworks are available for any version of Windows.
- Have installed the .NET Framework Redistributable package, which contains everything needed to run a program under the .NET Framework.

- ensure that the `Segue.SilkTest.Net.Shared.dll` has been installed into the Global Assembly Cache (GAC).
- set the Machine Zone Security.

After you complete these steps, you can use the Basic Workflow to test any .NET no-touch application.

Using the Open Agent

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.
7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

   Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.
8. Click Finish.

The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.

9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

    SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

    SilkTest opens the include file.
11. Record the test case whenever you are ready.

Using the Classic Agent

Tips for Working with .NET

This topic discusses common problems when testing .NET applications with SilkTest and how to solve them.

DefaultBaseState() enters an infinite loop when trying to close a dialog box

This problem is probably caused by the tag of the dialog box. If you declared the dialog box with multitags, make sure that the first tag in the multitag is correct. Usually the first tag is the caption of the dialog box. If the dialog box can have multiple captions, use wildcards to create a caption that covers all of them. Alternatively, if the dialog box has a valid window ID, you may want to use that as the tag or first multitag instead of the caption. If you must rely on a multitag for the dialog box, then be sure to close the dialog box explicitly as part of your test case.

This is not a problem for just the .NET extension; it can occur with any application. However, .NET applications are one of the few for which dialog boxes have valid Window IDs. Other dialog boxes only have valid captions, so they usually only have a single tag instead of a multitag. Therefore if the tag is wrong, it needs to be corrected in order to run the test case.

Predefined class definition file for .NET

The `dotnet.inc` file provides definitions for the supported .NET classes. Each class in the file lists the prototypes of all properties and methods for the class.

If you did not install .NET support, but later want to enable it to test your applications, you must manually edit `dotnet.inc` in order to use the predefined class definitions; otherwise, you will have to record all class definitions yourself.

DefaultBaseState does not work

After you remove or change the security string for the Framework, the DefaultBaseState might not work when you run SilkTest. Follow the instructions described in *Setting Your Machine Zone Security* and use the instructions that begin with *Open your Control Panel…*. Be sure to set the security for the Framework.

SilkTest only records the first character when you press and hold a key in a .NET text box or combobox

If the cursor is in a text box or a combobox and you enter text by pressing down and holding a key, so that multiple characters are entered, only the first character is recorded. For example, if you press and hold the "x" key causing many x's to display in the text box, SilkTest does not record "xxxxxxxxxxxxx"; SilkTest only records a single "x". You have to manually edit the argument to `SetText` in the script if you want SilkTest to play back multiple characters.

Enabling .NET Support

This functionality is available only for projects or scripts that use the Classic Agent.

Before testing .NET controls, you need to enable extensions.

Do you need to record additional classes?

- If your application contains only the standard .NET WinForms controls, then you do not need to record additional classes. If you are not sure, see .NET classes.
- If your application uses other controls, for example, third-party controls, and the Classic Agent, you must record classes for these additional controls. After you record the class, you can retrieve information about any number of instances (objects) of that class.

No-Touch Windows Forms Application Support

This functionality is available only for projects or scripts that use the Classic Agent.

Windows Forms applications are desktop applications that are built using the Windows Forms classes of the .NET Framework. The .NET Framework allows system administrators to deploy applications and updates to applications through a remote Web server. This technology is called no-touch deployment. With no-touch deployment, applications can be downloaded, installed, and run directly on the machines of the user without any alteration of the registry or shared system components.

No-touch deployment support is part of the .NET extension of SilkTest; this includes Window Forms applications.

SilkTest provides built-in support for testing .NET no-touch Windows Forms applications. SilkTest does not support side-by-side execution of no-touch applications.

Before you begin testing your no-touch application, see the *Prerequisites for Testing .NET No-Touch Applications*.

No-Touch Sample Application

You can use the .NET Test Application as a sample No-Touch application, by first copying it to a web server that has the .NET Framework installed. Then, open a browser on a different machine and type the link to the sample application. You do not need to install SilkTest on either machine to run the test application.

Recording New Classes for .NET Controls

This functionality is available only for projects or scripts that use the Classic Agent.

After you enable extensions for testing a .NET Windows Forms application with the Classic Agent, if you see CustomWin declarations in the Record Window Declarations, then click **Record** > **Class** in order to work with the CustomWin controls. The process of recording a class involves querying the objects in your application, retrieving information on properties and methods, and then translating the information into 4Test-style prototypes. SilkTest does this automatically when you click **Record** > **Class**.

When you use the **Record Class** dialog box to record new classes for .NET controls, SilkTest automatically inserts the __typeinfo keyword in front of any method that has parameters that are either:

- Of type POINT or RECT.
- Of a type that has been declared explicitly in 4Test and uses the alias mechanism.

We recommend that you create an include file, for example `userclass.inc`, for your new class definitions, instead of entering them in `dotnet.inc`. The `dotnet.inc` include file is shipped with SilkTest and we reserve the right to modify this file in future releases. If you modify `dotnet.inc`, you may have to integrate your changes into future versions of that file.

To record new classes:

1. Start the .NET standalone Windows Forms application.
2. Open the include file that you created for your new class definitions.
   For information on how to load class definition files, see *Several Ways to Load Class Definition Files*.
3. Click **Record** > **Class** > **Scripted** to open the **Record Class** dialog box.
4. Position the mouse pointer over the control for which you want to record a class.
5. When the correct name displays in the **Window** text box, press **Ctrl+Alt**.

Properties and methods for that class are displayed in the **Record Class** dialog box. Do not edit the tag name in the **Tag** text box. If you check **Show all methods** on the **Record Class** dialog box, you see a commented section called **Other methods**. Some methods are preceded by two slashes (//) and others are preceded by three slashes plus two asterisks (/// **). The methods that are preceded by "/// ** " cannot be called by SilkTest; they are included only for reference purposes.

The methods in the **Other methods** section that are preceded by two slashes can potentially be called by SilkTest. These are methods that:

- Have parameters or return values of a type not declared in 4Test. To call such a method, you must explicitly declare the types using alias. The data types in the argument list are just suggestions. You must consult the documentation for the control in order to determine the native data type name and definition. After you have declared the types, you should compile your frame file and click **Record** > **Class**. Now the recorder will recognize your new data types and will record as usable those methods that were previously commented out because the data types had not been defined.
- Are overloaded methods, in other words methods for which the parameter list may vary. 4Test does not support overloading of methods, since each method must have a unique name. You may choose to un-comment one of the overloaded methods for use within your test scripts.

6. Click the **Derived From** list box to see the list of available 4Test classes. Then proceed as follows:

   - If there is a class type available that maps directly to your object, choose it. For example, if your object is a SuperListBox, you would likely choose `ListBox`. Your object will inherit all the standard 4Test methods and properties defined for a list box.

     Note: Similarly named objects might not behave as expected.

   - If there is not a class type that maps directly to your object, choose `Control`, which is a generic class.

   See `winclass` declaration and derived class for more details.

   The Agent provides special handling for certain classes of objects. If your object is one of these types, but does not work correctly while you are testing your application, you will also need to class map the object after completing this procedure. For additional information, see *Options for Nongraphical Custom Controls*.
7. Click **Paste to Editor** to paste the new class into the include file.
8. Repeat this process for every type of control in your application that does not display in the list of classes provided.
9. When you are done recording classes, click **Close**.

Recording Actions on the DataGrid

This functionality is available only for projects or scripts that use the Classic Agent.

For standard `DataGrids`, SilkTest records the following 4Test methods using the Classic Agent, depending on the location or component in the grid:

- `ClickCell()`
- `ClickCellButton()`
- `ClickCol()`
- `ClickRow()`
- `Collapse()`
- `Expand()`
- `SetCellValue()`
- `SetFocusCell()`

Notes

For DataGrids with natively supported, embedded controls, SilkTest records the appropriate method call for the control with which you are interacting.

For DataGrids with embedded custom controls, SilkTest records low-level events, such as TypeKeys and Click.

SilkTest records a ClickCell before a SetCellValue.

If you use the SilkTest Open Agent, the `DataGrid` class uses a different set of methods.

Example

If you record changing the value "Pine" to "Maple" in the Cell "Last_Name", SilkTest generates:

```
SwfDialogBox("SamplesExplorer").SwfDialogBox("Sort").DataGrid("P
rototype Grid")
  .ClickCell ({{1,1,2}, "Last_Name"})
SwfDialogBox("SamplesExplorer").SwfDialogBox("Sort").DataGrid("P
rototype Grid")
  .SetCellValue({{1,1,2}, "Last_Name"}, "Maple")
```

Record Class, Window Declarations, and Window Identifiers

Record Window Declarations displays an instance of the `DataGrid` class when you cursor over the grid.

Record Class is not supported for components in the `DataGrid`, but it does record class on Controls, such as SwfTextField and CustomWin, in a DataGrid cell.

When you hover the cursor over a grid, Record Window Identifiers displays an instance of the `DataGrid` class. Record Window Identifiers does not display any components in the grid; it displays controls, such as SwfTextField and CustomWin, in a DataGrid cell.

Setting Your Machine Zone Security

This functionality is available only for projects or scripts that use the Classic Agent.

Before you can begin testing .NET no-touch applications, you must set up your machine security. You can disable security so that SilkTest can test your no-touch application using the command prompt or the control panel.

Setting security changes various permissions for .NET no-touch applications. For example, UIPermission controls access to user interface, ReflectionPermission controls access to metadata through the System.Reflection APIs, and SecurityPermission controls a set of security permissions applied to code.

Related topics
Setting Your Machine Zone Security using the Command Prompt

1. Open a command prompt.
2. Type `caspol -machine -chggroup LocalIntranet_Zone FullTrust`.
   This opens the specified zone to the Full Trust level.

`LocalIntranet_Zone` is just one example; you might need to adjust to a different code group, depending on what your application is using. Check with your application's developer if you are not sure.

Now that you have set the Machine Zone Security, make sure you have installed the `Segue.SilkTest.Net.Shared.dll`.

Setting Your Machine Zone Security using the Control Panel

1. Open the Control Panel and navigate to the **Administrative Tools** folder.

2. Double-click **Microsoft.NET Framework Configuration**.
3. In the **.NET Framework Configuration** tool, click the **Runtime Security Policy** node.
4. Click **Adjust Zone Security**.
5. Select **Make changes to this computer** and click **Next**.
6. On the **Adjust the Security Level for Each Zone** dialog box, choose **Full Trust level for this zone**.
7. Click **Next** to apply your changes.
8. Click **Finished**.
9. Repeat as necessary for the other Framework Configuration (if you have both installed).

Now that you have set the Machine Zone Security, make sure you have installed the `Segue.SilkTest.Net.Shared.dll`.

Ensuring that the Segue.SilkTest.Net.Shared.dll has been Installed

This functionality is available only for projects or scripts that use the Classic Agent.

Installing the SilkTest program should install `Segue.SilkTest.Net.Shared.dll` into the GAC. To verify that the SilkTest DLL is in the GAC:

1. Click **Control Panel** > **Administrative Tools**.
2. Click **Microsoft .NET Framework <version number> Configuration**.
3. Right-click **Assembly Cache** under **My Computer**.
4. Click **View** > **Assemblies** .
   The item Segue.SilkTest.Net.Shared should be in the list. You may have to scroll down to see it.
5. If the item Segue.SilkTest.Net.Shared is not in the list, add it by doing one of the following:
   - Right-click **Assembly Cache** under **My Computer** and click **Add**, then browse to the DLL in the SilkTest installation directory.
   - Click `Segue.SilkTest.Net.Shared.dll` and drag it into the `<Windows directory>\Assembly directory`, which will cause the DLL to be added to the GAC.

Now that you have installed the `Segue.SilkTest.Net.Shared.dll`, you must also set the machine zone security.

Suppressing Controls for Certain Classes

You can suppress the controls for certain classes for .NET, Java SWT, and Windows API-based applications. For example, you might want to ignore container classes to streamline your testcases. Ignoring these unnecessary classes simplifies the object hierarchy and shortens the length of the lines of code in your test scripts and functions. Container classes or ‘frames’ are common in GUI development, but may not be necessary for testing.

Common classes that are suppressed include:

.NET	Group
Java SWT	org.eclipse.swt.widgets.Composite
	org.eclipse.swt.widgets.Group
Windows API-based applications	Group

To suppress the controls for .NET applications, you must use the Classic Agent. To suppress the controls for Java SWT and Windows API-based applications, you must use the Open Agent.

1. Click **Options** > **Class Map**.
   The **Class Map** dialog opens.
2. In the Custom class field, type the name of the class that you want suppress.

The class name depends on the technology and the extension that you are using. For Windows API-based applications, use the Windows API-based class names. For Java SWT applications, use the fully qualified Java class name.

To ignore the **SWT_Group** in a Windows API-based application, type `SWT_Group`. For Java SWT applications, type `org.eclipse.swt.widgets.Group`.

3. In the **Standard class** list, select **Ignore**.
4. Click **Add**. The custom class and the standard class display at the top of the dialog.

Infragistics Controls

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest has several built-in 4Test classes that support recording and playback for key Infragistics Window Forms controls. These include:

- The UltraWinGrid controls through the 4Test `DataGrid` class.
- The UltraWinToolbars container and the elements within the UltraWinToolbars through the 4Test `Toolbar` class (Infragistics' ToolBase).

Native support means that features such as the following are available when testing these controls:

- Action-based recording.
- Record Window declarations.
- Record identifiers.
- Record class.

Before you can use SilkTest to test Infragistics Windows Forms controls, you must:

- Install the .NET Framework.
- Install NetAdvantage Window Forms.

For specific versions of these applications and other installation requirements, refer to the *Release Notes*.

Testing the Infragistics UltraWinGrid and UltraWinToolbars

SilkTest has two classes for Infragistics support:

- `DataGrid`, which supports the Infragistics UltraWinGrid.
- `UltraWinToolbars`, which supports the Infragistics UltraWinToolbars container and the UltraWinToolbars elements within the UltraWinToolbars.

There are many 4Test methods available with these new classes, some of which are available for recording.

Support for Infragistics .dll Files

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest supports the following Infragistics .dll files (assemblies):

- Infragistics.Win.UltraWinGrid
- Infragistics.Win
- Infragistics.Shared
- Infragistics.Win.UltraWinToolbars

SilkTest supports the following Infragistics NetAdvantage versions:

- CLR 1.1:
- • 4.3.20043.27
  - 5.1.20051.37
  - 5.2.20052.27
  - 5.3.20053.50

    - 6.1.20061.28
    - 6.2.20062.34
    - 6.3.20063.53
- CLR 2.0:
- • 7.3.20073.38

The CLR 2.0 library differs from the CLR 1.1 library. As a result, .dll files cannot be shared between CLR 2.0 and 1.1.

For up-to-date information about supported versions, refer to the *Release Notes* in **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

If you are testing an application that uses a different version of these .dll files, you must modify either the application configuration or the machine configuration file with instructions that redirect SilkTest to use the version that your application supports. SilkTest can be configured post-installation to support NetAdvantage version variations in the last two version fields, for example, x.x.20061.28.

If you do not redirect SilkTest to use the proper .dll files, you may experience problems with SilkTest unable to record Infragistics controls correctly, such as the following:

- Recording `Typekeys`() on a `DataGrid`, instead of correctly recording `ClickCell`() and `SetCell`().
- Not recognizing the buttons inside the Toolbar when your mouse is pointing to the Toolbar.

Notes

- Since applications can specify which specific directory is used to load .dlls, the binding method described below will work only if it is not overwritten or ignored.
- There may be other backward compatibility issues with Infragistics .dlls. For more details about these issues, refer to the documentation of your .NET Framework SDK.

To redirect the .dlls, you must:

1. Choose whether to configure the machine or the application.
2. Choose whether to configure by editing a file or through the Control Panel. You can use either of these methods to configure the machine or the application.

Configuring .NET

This functionality is available only for projects or scripts that use the Classic Agent.

Now that you have decided whether to change the configuration for just your application under test or for your entire machine, you must decide how you want to make the change. You can use either of these methods, regardless of whether you are changing the configuration for the application or for the machine:

- Edit a configuration file directly.
- Use the Control Panel to edit the configuration file.

Choosing to Configure the Machine or the Application

This functionality is available only for projects or scripts that use the Classic Agent.

You may edit the configuration file for just your application or you may edit the configuration file for your entire machine. If you do not have a configuration file, you may create one; see the sample file if you are interested in creating one.

If you want to change the application configuration, you need to do so on the machine where the application .exe is, in the same directory as that .exe. This approach establishes a "default" configuration that is associated with the application.

If you want to change the machine configuration, you need to do so on the machine where the application will run (which is the same location as the SilkTest Agent). This approach establishes a configuration that will override any application configuration that may exist.

In either case, you can use the direct file editing or the Control Panel to edit the configuration file.

Recording Actions on the Infragistics Toolbars

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest records actions on the Infragistics UltraWinToolbars with a prefix of "Toolbar". All other components, except for .NET SWF controls, are mapped as a CustomWin.

Class	SilkTest records	Description
`ToolbarButton`	`Click()`	You click a button on the toolbar.
`ToolbarComboBox`	`Select()`	You choose an item from the list box.
	`SetText()`	You type in the text area.
`ToolbarList`	`Select()`	You select an item.
	`Unselect()`	You unselect an item.
`ToolbarPopup`	`Select()`	You selected a new item.
`ToolbarPopupMenu`	`DropDown()`	You interact with the menu.
`ToolbarTextBox`	`SetText()`	You type in the text area.

Record Class, Window Declarations, and Window Identifiers

SilkTest records the methods and properties of any UltraWinToolbars control or component when you Record/Class/Scripted on children of the `UltraWinToolbars` class. Likewise, Record Window Declarations displays the `UltraWinToolbars` class declaration and all the supported components and controls that are children of the UltraWinToolbars control.

When you hover the cursor over a component or a control in UltraWinToolbars, Record Window Identifiers displays that component or control as a child of an instance of the `UltraWinToolbars` class.

WPF Applications

Overview of WPF Application Support

SilkTest provides built-in support for testing Windows Presentation Foundation (WPF) applications using the Open Agent. SilkTest supports standalone WPF applications and can record and play back controls in .NET version 3.5 or later.

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

Sample Applications

SilkTest provides several sample applications for WPF. You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, click **StartProgramsSilkSilkTest <version>Sample ApplicationsMicrosoft .NET** to select the sample application that is right for your environment.

Object Recognition

WPF applications support hierarchical object recognition and dynamic 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.

When you record a test case with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Existing test cases that use dynamic object recognition without locator keywords in an INC file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object recognition without locator keywords in an INC file.

Supported Controls

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

If you have an existing project that includes scripts that use the earlier MSUIA technology domain, the test cases will continue to work. However, if you use OPT files with the earlier MSUIA technology domain, manual changes are necessary to run the test cases successfully.

You can also set up your test cases to use the deprecated MSUIA technology domain and the new WPF technology domain side-by-side. This enables you to migrate to the new WPF technology domain because both the new WPF and deprecated MSUIA locators display in the Locator Spy. You can then use the Locator Spy to migrate scripts to the new technology domain on a step-by-step basis. For example, you can change an application's main menu that was identified by `//MsuiaWindow//MsuiaMenu` previously to `//WPFWindow//WPFMenu`. You can enable side-by-side functionality by manually changing your OPT file, if you use option files rather than project files, or the `projectname.ini` file.

Microsoft UI Automation (MSUIA) is used to automate WPF applications. For a complete list of the controls available for WPF testing, see the *WPF Class Reference*. For a complete list of the deprecated controls, see the *MSUIA Class Reference*.

Supported methods and properties for MSUIA controls depend on the actual implementation and runtime state. The methods and properties may differ from the list that is defined for the corresponding 4test class. To determine the methods and properties that are supported in a specific situation, use the `SupportedMethods` and `SupportedProperties` methods.

The caption value is mapped to the `Name` property by MSUIA.

For additional information abut MSUIA, refer to MSDN.

SilkTest Agent Support

When you create a new WPF project, SilkTest uses the Open Agent by default. However, you can use both the Open Agent and the Classic Agent within a single WPF environment. Certain functions and methods run on the Classic Agent only. As a result, if you are running an Open Agent project, the Classic Agent may also open because a function or method requires the Classic Agent.

Attributes for Windows Presentation Foundation (WPF) 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 WPF applications include:

- *automationId*
- *caption*
- *className*
- *name*

Note: Attribute names are case sensitive.

Dynamic Object Recognition

To identify components within WPF scripts, you can specify the *automationId*, *caption*, *className*, or *name*. The name that is given to an element in the application is used as the *automationId* attribute for the

locator if available. As a result, most objects can be uniquely identified using only this attribute. For example, a locator with an *automationId* might look like: `//WPFButton[@automationId='okButton']`.

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

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

During recording, SilkTest 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*, SilkTest uses the *automationId* when creating the locator.

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

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

Classes that Derive from the WPFItemsControl Class

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

Working with the control	Most controls contain methods and properties for typical use cases. The items are identified by text or index.

For example:

```
listBox.Select("Banana")
listBox.Select(2)
tree.Expand("/Fruit/Banana")
```

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

Custom WPF Controls

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

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

Deriving classes from UserControl	This is a typical way to create compound controls. SilkTest recognizes these user controls as `WPFUserControl` and provides full support for the contained controls.
Deriving classes from standard WPF controls, such as ListBox	SilkTest treats these controls as an instance of the standard WPF control that they derive from. Record, playback, and recognition of children may not work if the user control behavior differs significantly from its base class implementation.
Using standard controls that use templates to change their visual appearance	Low-level replay might not work in certain cases. Switch to high-level replay in such cases.

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

Setting WPF Classes to Expose During Recording and Playback

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

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

1. Click **Options** > **Recorder**.
   The **Recording 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**.

Dynamically Invoking WPF Methods

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

Call dynamic methods on WPF objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a WPF control, use the `GetDynamicMethodList`() method.

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

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty`() method. To retrieve a list of supported dynamic properties for a WPF control, use the `GetPropertyList`() method.

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

Supported Methods and Properties

The following methods and properties can be called:

- Methods and properties that SilkTest supports for the WPF control.
- All public methods and properties that the MSDN defines for the WPF control.

  For example, for a `WPFListBox` object type, you can call all methods and properties that MSDN defines for the `System.Windows.Controls.ListBox` type.
- If the WPF control is a custom control that is derived from a standard WPF control, all methods and properties from the standard control can be called.

Supported Parameter Types

The following parameter types are supported:

- All built-in SilkTest types.

  SilkTest types includes primitive types (such as boolean, int, string), lists, and other types (such as Point and Rect).
- Enum types.
- Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visibility` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
- .NET structs and objects.

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

  WPF control parameters can be passed as WINDOW.

Returned Values

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

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

- A string for all other types.

  Call `ToString`() on returned .NET objects to retrieve the string representation.

  Example

  A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

  ```
  customControl.Invoke("Reset")
  REAL sum = customControl.DynamicInvoke("Add",{1,2})
  ```

  The calculator control also has a `LastCalculationResult` property. You can use the following code to read the property:

  ```
  REAL lastResult =
  customControl.GetProperty("LastCalculationResult")
  ```

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.

7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

   Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.
8. Click Finish.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

    SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

    SilkTest opens the include file.
11. Record the test case whenever you are ready.

Manually Updating .INI Files to Include the Deprecated WPF TechDomain

If you have an existing project that includes scripts that use the earlier, deprecated MSUIA technology domain, the test cases will continue to work. When you record new test cases for the existing project, the new WPF technology domain is used. You can also set up your test cases to use the deprecated MSUIA technology domain and the new WPF technology domain side-by-side. This enables you to migrate your existing scripts to the new WPF technology domain because both the new WPF and deprecated MSUIA locators display in the Locator Spy.

1. Open the `projectname.ini` file.

   By default, this file is located in `C:\Program Files\Silk\SilkTest\Projects\projectname`.
2. Navigate to the `TechDomains` entry.
3. Specify `TechDomains=WIN32, MSUIA, WPF`.

   When both technology domains are enabled, the top-level window is recognized as `WPFWindow`. This `WPFWindow` contains all the controls that are recognized by the by new WPF technology domain. The `WPFWindow` has a child control of the type `MSUIAWindow` that contains the controls that the deprecated MSUIA technology domain recognizes for this window. You can then use the Locator Spy to migrate scripts to the new technology domain on a step-by-step basis. For example, you can change an application's main menu that was identified by `//MsuiaWindow//MsuiaMenu` previously to `//WPFWindow//WPFMenu`.
4. Save the file.

Manually Updating .OPT Files to Use the Deprecated WPF TechDomain

If you have an existing project that includes scripts that use the earlier, deprecated MSUIA technology domain, the test cases will continue to work. However, if you use .OPT files with the earlier MSUIA technology domain, manual changes are necessary to run the test cases successfully. Otherwise, during replay the exception `E_WINDOW_NOT_FOUND` occurs and the Locator Spy does not display correctly when moving around the application that you are testing.

If you use project files, your test cases will continue to run successfully and this procedure is not required.

1. Navigate to the `TechDomains` entry in the .OPT file.

2. To change the WPF reference to use the deprecated MSUIA technology domain only, specify `TechDomains=WIN32, MSUIA`.

   For example, if the .OPT file contains `TechDomains=WIN32, WPF`, change it to `TechDomains=WIN32, MSUIA`.
3. To use the deprecated MSUIA technology domain side-by-side with the new WPF technology domain, specify `TechDomains=WIN32, MSUIA, WPF`.

   This enables you to migrate to the new WPF technology domain because both the new WPF and deprecated MSUIA locators display in the Locator Spy. When both technology domains are enabled, the top-level window is recognized as `WPFWindow`. This `WPFWindow` contains all the controls that are recognized by the by new WPF technology domain. The `WPFWindow` has a child control of the type `MSUIAWindow` that contains the controls that the deprecated MSUIA technology domain recognizes for this window. You can then use the Locator Spy to migrate scripts to the new technology domain on a step-by-step basis. For example, you can change an application's main menu that was identified by `//MsuiaWindow//MsuiaMenu` previously to `//WPFWindow//WPFMenu`.
4. Save the file.

MSUIA Class Reference

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

For a list of the current WPF classes, see the *WPF Class Reference*.

SilkTest includes the following 4Test classes for MSUIA support:

Note: These classes are deprecated.

- `MsuiaButton`
- `MsuiaCalendar`
- `MsuiaCheckBox`
- `MsuiaComboBox`
- `MsuiaCustom`
- `MsuiaDataGrid`
- `MsuiaDataItem`
- `MsuiaDocument`
- `MsuiaEdit`
- `MsuiaGroup`
- `MsuiaHeader`
- `MsuiaHeaderItem`
- `MsuiaHyperlink`
- `MsuiaImage`
- `MsuiaList`
- `MsuiaListItem`
- `MsuiaMenu`
- `MsuiaMenuBar`
- `MsuiaMenuItem`
- `MsuiaPane`
- `MsuiaProgressBar`
- `MsuiaRadioButton`
- `MsuiaScrollBar`
- `MsuiaSeparator`

- `MsuiaSlider`
- `MsuiaSpinner`
- `MsuiaSplitButton`
- `MsuiaStatusBar`
- `MsuiaTab`
- `MsuiaTabItem`
- `MsuiaTable`
- `MsuiaText`
- `MsuiaThumb`
- `MsuiaTitleBar`
- `MsuiaToolBar`
- `MsuiaToolTip`
- `MsuiaTree`
- `MsuiaTreeItem`
- `MsuiaWindow`

You can also view a list of the supported MSUIA controls in the `Msuia.inc` file. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\MSUIA\Msuia.inc`.

Microsoft UI Automation Exception Values

Microsoft UI Automation (MSUIA) is used to automate WPF applications. Under given error conditions, SilkTest generates exception values. The exception values for Microsoft UI Automation (MSUIA) include:

Exception	Description
E_MSUIA_BASE	A general exception, which is thrown when no other known exception occurs in MSUIA. Additional information displays in the exception text.
E_MSUIA_CONTROLPATTERN_NOT_SUPPORTED	An error occurred at runtime because the specified control pattern is not supported. For example, if you specify an invoke pattern for a button control when only a toggle pattern is supported, this error is displayed. For additional information about control patterns, refer to MSDN.

WPF Class Reference

When you configure a WPF application, SilkTest Classic automatically provides built-in support for testing standard WPF controls.

Silverlight Applications

Overview of Microsoft Silverlight Application Support

Microsoft Silverlight (Silverlight) is an application framework for writing and running rich internet applications, with features and purposes similar to those of Adobe Flash. The run-time environment for Silverlight is available as a plug-in for most web browsers.

SilkTest Classic provides built-in support for testing Silverlight applications. SilkTest Classic supports Silverlight applications that run in a browser as well as out-of-browser and can record and play back controls in Silverlight 3 (Silverlight Runtime 4) or Silverlight 4 (Silverlight Runtime 4).

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

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

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

Object Recognition

Silverlight applications support dynamic object recognition. You can create tests for dynamic object recognition in your test environment.

When you record a test case with the Open Agent, SilkTest Classic creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Supported Controls

SilkTest Classic includes record and replay support for Silverlight controls.

For a complete list of the controls available for Silverlight testing, see the *Silverlight Class Reference*.

Agent Support

When you create a Silverlight project, the Open Agent is assigned as the default agent.

Prerequisites

The support for testing Silverlight applications in Microsoft Windows XP requires the installation of Service Pack 3 and the Update for Windows XP with the Microsoft User Interface Automation that is provided in Windows 7. You can download the update from *http://www.microsoft.com/download/en/details.aspx?id=13821*.

Note: The Microsoft User Interface Automation needs to be installed for the Silverlight support. If you are using a Windows operating system and the Silverlight support does not work, you can install the update with the Microsoft User Interface Automation, which is appropriate for your operating system, from *http://support.microsoft.com/kb/971513*.

Locator Attributes for Identifying Silverlight Controls

Supported locator attributes for Silverlight controls include:

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

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

Dynamic Object Recognition

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	`//SLButton[@automationId="okButton"]`

Attribute Type	Description	Example
	developer uses this ID to identify the control in the application code.	
caption	The text that the control displays. When testing a localized application in multiple languages, use the *automationId* or *name* attribute instead of the *caption*.	`//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 SilkTest 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, SilkTest Classic 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 a *automationId* and a *name*, SilkTest Classic uses the *automationId* when creating the locator.

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

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

Dynamically Invoking Silverlight Methods

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

Call dynamic methods on Silverlight objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a Silverlight control, use the `GetDynamicMethodList`() method.

Call multiple dynamic methods on objects with the `DynamicInvokeMethods` method. To retrieve a list of supported dynamic methods for a Silverlight control, use the `GetDynamicMethodList`() method.

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty`() method. To retrieve a list of supported dynamic properties for a Silverlight control, use the `GetPropertyList`() method.

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

Supported Parameter Types

All built-in SilkTest Classic types	SilkTest Classic types include primitive types, for example boolean, int, and string, lists, and other types, for example Point and Rect.
Enum types	Enum parameters must be passed as string. The string must match the name of an enum value. For example, if the method expects a parameter of the .NET enum type `System.Windows.Visiblity` you can use the string values of `Visible`, `Hidden`, or `Collapsed`.
.NET structs and objects	Pass .NET struct and object parameters as a list. The elements in the list must match one constructor for the .NET object in the test application. For example, if the method expects a parameter of the .NET type `System.Windows.Vector`, you can pass a list with two integers. This works because the `System.Windows.Vector` type has a constructor with two integer arguments.
Other controls	Control parameters can be passed as `TestObject`.

Returned Values

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

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

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

Example

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

```
tabItem.DynamicInvoke("SelectionItemPattern.Select")
mySilverligtObject.GetProperty("IsPassword")
```

Scrolling in Silverlight

SilkTest Classic provides two different sets of scrolling-related methods and properties, depending on the Silverlight control.

- The first type of controls includes controls that can scroll by themselves and therefore do not expose the scrollbars explicitly as children. For example combo boxes, panes, list boxes, tree controls, data grids, auto complete boxes, and others.
- The second type of controls includes controls that cannot scroll by themselves but expose scrollbars as children for scrolling. For example text fields.

This distinction in SilkTest Classic exists because the controls in SilkTest Classic implement scrolling in those two ways.

Controls that support scrolling

In this case, scrolling-related methods and property are available for the control that contains the scrollbars. Therefore, SilkTest Classic does not expose scrollbar objects.

Examples

The following command scrolls a list box to the bottom:

```
listBox.SetVerticalScrollPercent(100)
```

The following command scrolls the list box down by one unit:

```
listBox.ScrollVertical(ScrollAmount.SmallIncrement)
```

Controls that do not support scrolling

In this case the scrollbars are exposed. No scrolling-related methods and properties are available for the control itself. The horizontal and vertical scrollbar objects enable you to scroll in the control by specifying the increment or decrement, or the final position, as a parameter in the corresponding API functions. The increment or decrement can take the values of the ScrollAmount enumeration. For additional information, refer to the Silverlight documentation. The final position is related to the position of the object, which is defined by the application designer.

Examples

The following command scrolls a vertical scrollbar within a text box to position 15:

```
textBox.SLVerticalScrollBar().ScrollToPosition(15)
```

The following command scrolls a vertical scrollbar within a text box to the bottom:

```
textBox.SLVerticalScrollBar().ScrollToMaximum()
```

Troubleshooting when Testing Silverlight Applications

SilkTest Classic cannot see inside the Silverlight application and no green rectangles are drawn during recording

The following reasons may cause SilkTest Classic to be unable to see inside the Silverlight application:

Reason	Solution
You use a Mozilla Firefox version prior to 4.0.	Use Mozilla Firefox 4.0 or later.
You use a Silverlight version prior to 3.	Use Silverlight 3 (Silverlight Runtime 4) or Silverlight 4 (Silverlight Runtime 4).

Your Silverlight application is running in windowless mode.	SilkTest Classic does not support Silverlight applications that run in windowless mode. To test such an application, you need to change the Web site where your Silverlight application is running. Therefore you need to set the `windowless` parameter in the object tag of the HTML or ASPX file, in which the Silverlight application is hosted, to `false`.
	The following sample code sets the `windowless` parameter to `false`:
	`<object ...>`
	`  <param name="windowless" value="false"/>`
	`  ...`
	`</object>`

Silverlight Class Reference

When you configure a Silverlight application, SilkTest Classic automatically provides built-in support for testing standard Silverlight controls.

Testing ActiveX/VB controls

Overview of ActiveXVisual Basic Support

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Visual Basic 5 and 6

SilkTest provides built-in support for testing ActiveX controls and Visual Basic 5 and 6 native controls. These controls can be embedded in:

- Visual Basic 5 and 6 applications
- Other 32-bit Windows applications
- HTML Web pages

In addition, you can test more than one application at a time.

Visual Basic 4

If you are testing Visual Basic 4 applications, you do not have access to properties and methods in native controls, just ActiveX controls. Since most Visual Basic 4 native controls map to Windows native controls, you can use SilkTest's class mapping feature to test native controls.

Before you get started testing, see: *Assumptions and Terminology*.

Where to go from here

Read these topics in the following order:

1. Enabling ActiveX/Visual Basic support
2. Predefined classes for ActiveX/Visual Basic controls
3. Recording new classes for ActiveX/Visual Basic controls, if necessary
4. Testing ActiveX/Visual Basic controls

Enabling ActiveXVisual Basic support

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Before testing Visual Basic and ActiveX controls in a stand-alone application or in Internet Explorer, you need to enable extensions. If you are testing ActiveX controls in Internet Explorer, you must complete Set up for testing ActiveX controls or Java applets in the browser.

Predefined classes for ActiveXVisual Basic controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

In addition to the 4Test classes provided in SilkTest, support for Visual Basic with ActiveX controls includes predefined class definitions for:

- The native Visual Basic controls included in Microsoft Windows Visual Basic 6.0 Professional Edition.
- The ActiveX controls bundled with Visual Basic 6.0 Professional Edition.

These definitions are provided in a file as a convenience to help you quickly get started testing your applications.

Property names that begin with the prefix VB, for example, `rVBHeight` of the `OLEAniPushButton` class, are available only in Visual Basic applications. These properties are added to an ActiveX control by the Visual Basic environment. They are not available in C/C++ environments.

Do you need to record additional classes?

If your application contains only those controls shipped with the Microsoft Windows Visual Basic Professional Edition, then you do not need to record additional classes. If you are not sure, review the controls in your application and compare them to the table in the list of controls.

- If your application uses only these types of controls, you do not need to record additional classes. Go to *Testing ActiveX/Visual Basic controls*.
- If your application uses controls other than those listed in the table, for example, third-party ActiveX controls, you must record classes for these additional controls, as described in Recording new classes for ActiveX/Visual Basic controls. After you record the class, you can retrieve information about any number of instances (objects) of that class.

Predefined class definition file for VB

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The `vbclass.inc` file provides 4Test class definitions for the native Visual Basic controls and bundled ActiveX controls supported in included in Microsoft Windows Visual Basic 6.0 Professional Edition. Each class in the file lists the prototypes of all properties and methods for the class.

If you did not install Visual Basic/ActiveX support but later want to enable it to test your applications, you must manually edit `startup.inc` in order to use the predefined class definitions; otherwise, you will have to record all class definitions yourself. The procedure for manually enabling Visual Basic/ActiveX support is described in Enabling the ActiveX/Visual Basic extension.

List of predefined ActiveXVisual Basic controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The following table lists each of the ActiveX and native Visual Basic controls for which classes are provided, the enhanced Visual Basic- and ActiveX-specific 4Test class each control is associated with, and the standard 4Test class from which that class is derived. For example, the `3D Check Box` control is associated with the `OLESSCheck` class, which is derived from the 4Test class `CheckBox`.

Native Visual Basic or ActiveX control	4Test class for the control	Standard 4Test class derived from
3D Check Box Control	OLESSCheck	CheckBox
3D Command Button Control	OLESSCommand	PushButton
3D Frame Control	OLESSFrame	StaticText
3D Option Button Control	OLESSOption	RadioButton
3D Panel Control	OLESSPanel	Control
3D Group Push Button Control	OLESSRibbon	Control
Animation Control	OLEAnimation	Control
Animated Button Control	OLEAniPushButton	PushButton
CheckBox Control	VBCheckBox	CheckBox
ComboBox Control	VBComboBox	ComboBox
CommandButton Control	VBCommandButton	PushButton

Native Visual Basic or ActiveX control	4Test class for the control	Standard 4Test class derived from
Data Control	VBData	Control
DBCombo Control	OLEDBCombo	ComboBox
DBGrid Control	OLEDBGrid	Control
DBList Control	OLEDBList	ListBox
DirListBox Control	VBDirListBox	ListBox
DriveListBox Control	VBDriveListBox	PopupList
FileListBox Control	VBFileListBox	ListBox
Form	VBMainForm	MainWin
Form	VBChildForm	ChildWin
Form	VBForm	DialogBox
Frame Control	VBFrame	StaticText
Gauge Control	OLEGauge	Control
Graph Control	OLEGraph	ControlMultiWin
Grid Control	OLEGrid	Control
HScrollBar Control	VBHScrollBar	ScrollBar
Image Control	VBImage	Control
Key State Control	OLEMhState	Control
Label Control	VBLabel	StaticText
ListBox Control	VBListBox	ListBox
ListView Control	OLEListView	ListView
Masked Edit Control	OLEMaskEdBox	TextField
MDIForm	VBMDIForm	MainWin
MSChart Control	OLEMSChart	Control
MSFlexGrid Control	OLEMSFlexGrid	Control
Multimedia MCI Control	OLEMMControl	ControlMultiWin
OLE Container Control	VBOLE	Control
OptionButton Control	VBOptionButton	RadioButton
Outline Control	OLEOutline	Control
PictureBox Control	VBPictureBox	Control
ProgressBar Control	OLEProgressBar	Control
RichTextBox Control	OLERichTextBox	TextField
Shape Control	VBShape	Control
Slider Control	OLESlider	Scale
SpinButton Control	OLESpinButton	Control
SSTab Control	OLESSTab	PageList

Native Visual Basic or ActiveX control	4Test class for the control	Standard 4Test class derived from
StatusBar Control	OLEStatusBar	StatusBar
TabStrip Control	OLETabStrip	Control
TextBox Control	VBTextBox	TextField
Toolbar Control	OLEToolbar	ToolBar
TreeView Control	OLETreeView	TreeView
UpDown Control	OLEUpDown	UpDown
VScrollBar Control	VBVScrollBar	ScrollBar

Access to VBOptionButton control methods

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To access the ActiveX methods and properties of a control of class `VBOptionButton`, you must set the **Don't Group Radio Buttons Into a List** option in the Agent Options dialog. There are several ways to do this:

Locally Enter the following statement in your script(s): `Agent.SetOption (OPT_RADIO_LIST, FALSE)`. The advantage of setting the option locally is that you can still treat a group of buttons as a radio list, for example, for selection purposes.

Globally Open the **Agent Options** dialog, click the **Compatibility** tab, and check the **Don't Group Radio Buttons into a List** check box.

0-based arrays

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When you access arrays using the methods provided in the Visual Basic and ActiveX extension, the arrays are 0-based; that is, the first value is stored in slot 0. In contrast, `GetArrayProperty` and `SetArrayProperty`, two methods provided for backward compatibility with previous releases, used 1-based arrays. The following example illustrates the current and old syntax:

```
testcase GetColWidthForGrid () appstate none

   INTEGER iWidth1, iWidth2

   //Get width of col. 1 in MyGrid, using current syntax
   //Note that index (passed to GetColWidth method) is 0
   iWidth1 = MyApp.MyGrid.GetColWidth(0)

   //Changes the width of col. 1, using current syntax
   MyApp.MyGrid.SetColWidth (0, 555)

   //Gets width of col. 1 in MyGrid, using old syntax
   //Note that index of the ColWidth property arrow
   //(passed to GetArrayProperty method) is 1
   iWidth2 = MyApp.MyGrid.GetArrayProperty ("ColWidth", 1)
```

Passing an index of 0 to `GetArrayProperty` causes an error at runtime.

Dependent objects and collection objects

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Active X controls can be composed of objects which themselves expose properties and methods. An example is the Data control, which contains a Recordset control. Such contained objects are often referred to as dependent objects because they don’t exist outside the context of the containing control. In many cases, dependent objects are arranged into groups, or collections. For example, a TreeView control contains a collection of Node objects.

Users of the ActiveX control need a way to get at dependent objects. In the case of a simple dependent object, the outer control typically exposes a property that provides access to the contained object. In the case of a collection, the outer control provides access to the items in a collection through an intermediate object called a collection object.

You can get programmatic access to dependent objects by having the relevant control class inherit from a special class provided for this purpose: the `CompoundControl` class. This class provides methods for accessing the properties and methods of a simple dependent object, a collection object, or the individual items within a collection.

Working with dynamically windowed controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

In Internet Explorer, ActiveX controls may be dynamically windowed, which means their windows may come and go, or change location as the object is scrolled in and out of view. Consequently, recording declarations and actions against such objects can be tricky. You can achieve the most consistent results by bringing the ActiveX control into full view when recording declarations, classes, or actions. If you don't bring the ActiveX control into full view, SilkTest might not recognize it correctly.

Window timeout

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When you install Visual Basic and ActiveX support in SilkTest, the **Window Timeout** setting in the **Agent Options** dialog is initially set to 20 seconds. This setting determines how long the recovery system waits when checking to see if your application exists. If you choose to change the **Window Timeout** setting to a low number or 0, you may encounter a timing problem, where rapidly exiting and restarting an application may generate "Windows not found" errors.

Setting this option is not required when testing ActiveX controls in Internet Explorer, but is recommended as a general practice.

Conversion of BOOLEAN values

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

In some cases, when you record a class declaration, the result is a method prototype that takes a SHORT data type as a parameter even though the associated Visual Basic property takes a BOOLEAN parameter.

For example, in the following Visual Basic prototype, the syntax of the `TabEnabled` property of the `SSTab` class is:

```
object.TabEnabled (tab) [= boolean]
```

where boolean, the return value, is a BOOLEAN value: TRUE for enabled, FALSE for disabled. The prototype for its method equivalent in the SSTab class is shown in `vbclass.inc` as:

```
ole VOID SetTabEnabled (SHORT Index, SHORT Arg1)
```

Notice that the `SetTabEnabled` method takes a value of type SHORT as its second argument, which is equivalent to the BOOLEAN argument of the property. However, because 4Test and ActiveX define the BOOLEAN type differently, attempting to pass TRUE or FALSE as you would in Visual Basic will generate an argument type mismatch error in SilkTest. Use the following constant values instead, which are defined in `oleclass.inc`:

- `OLE_TRUE (which equals –1)`
- `OLE_FALSE (which equals 0)`

Control tests4Test versus ActiveX methods

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

The most effective way to test a control depends on the type of the control. For some controls, the inherited 4Test methods are more efficient; for others, the ActiveX methods or properties are more efficient. In general, we recommend that you begin by trying the 4Test methods associated with the class from which the control is derived. For example, for the `OLERichTextBox` use the 4Test methods and properties for `TextField`.

Control access is similar to Visual Basic

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can access a control's internal properties and methods using the dot operator and a syntax similar to standard Visual Basic. For example, to retrieve the number of rows in a Grid ActiveX control called MyGrid you might use the `iRows` property, as follows:

```
INTEGER iNumRows
iNumRows = MyApp.MyGrid.iRows
```

Another example: to set the number of rows in the same grid to 10, you might use:

```
MyApp.MyGrid.iRows = 10
```

Testing ActiveXVisual Basic controls

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

You are ready to begin testing your application. First, please read the information on designing and recording testcases in *About testcases*. Become familiar with the basic concepts of testcase design, including the setup, verification, and cleanup stages, and the **Record** > **Testcase** and **Record** > **Action** menu items.

The following topics contain additional information about testing your application.

- Access to OptionButton control methods
- Accessing controls is similar to Visual Basic
- Control tests: 4Test versus ActiveX methods
- Conversion of BOOLEAN values
- Dependent Objects and Collection Objects
- Extending classes
- Ignore an ActiveX class
- Methods for accessing indexed properties: Set and Get
- Naming conflicts
- Recognizing window hierarchy in the ActiveX extension
- Returned STRING values
- Tag declaration for SSTab control
- Visual Basic and ActiveX exception values
- Window timeout
- Working with dynamically windowed controls
- 0-based arrays

ActiveXVisual Basic exception values

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest generates exception values under given error conditions. These values are described in the online Help. In addition, the ActiveX/Visual Basic support defines the following set of exception values:

E_SPY_NOT_RESPONDING	Unable to connect to the Visual Basic or Windows application with embedded OLE controls.
E_OBJ_CALL_FAILED	The method or property call returned an error.
E_OBJ_NOT_FOUND	The Visual Basic or OLE control object could not be found.
E_ARG_TYPE_MISMATCH	One of the arguments has the wrong type.
E_BAD_ARG_COUNT	Wrong number of arguments for this method or property call.
E_ARG_VAL_OUT_OF_RANGE	One of the arguments had a value that was out of range.

Recording new classes for ActiveXVisual Basic controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The process of recording a class involves querying the objects in your application, retrieving information on properties and methods, and then translating the information into 4Test-style prototypes. SilkTest does this automatically for you when you select **Record** > **Class**.

We recommend that you create an include file (for example, `userclass.inc`) for your new class definitions, instead of entering them in `vbclass.inc`. This way you will not have to update `vbclass.inc` each time you record new class definitions.

1. Start your application. If you are recording classes for controls on a web page, navigate to that Web page.
2. Open the include file you created for your new class definitions. For information on how to load class definition files, see *Several ways to load class definition files*.
3. Click **Record** > **Class** > **Scripted** to open the Record Class dialog.
4. Position the mouse pointer over the control for which you want to record a class.
5. When the correct name appears in the Window field, press `Ctrl+Alt`. Properties and methods for that class are displayed in the **Record Class** dialog. Do not edit the tag name in the Tag field.
6. Click the **Derived From** drop-down menu to see the list of available 4Test classes. Then proceed as follows:

   - If there is a class type available that maps directly to your object, choose it. For example, if your object is a `SuperListBox`, you might choose `ListBox` (note that similarly named objects might not behave as expected). Your object will inherit all the standard 4Test methods and properties defined for a list box.
   - If there is not a class type that maps directly to your object, choose **Control**, which is a generic class.

   See winclass declaration and derived class for more details.

   The Agent provides special handling for certain classes of objects. If your object is one of these types, but does not work correctly while you are testing your application, you will also need to class map the object after completing this procedure. For more information, see *Options for nongraphical, custom controls*.
7. Click **Paste to Editor** to paste the new class into the include file.

8. Repeat this process for every type of control in your application that does not appear in the list of classes provided. When you are done recording classes, and then click **Close**.

Disabling ActiveXVisual Basic support

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You disable ActiveX/Visual Basic support for a particular application on the host machine, rather than in general.

1. On the host machine, click **Options** > **Extensions** to open the **Extensions** dialog.
2. For the application you want to disable, uncheck the **ActiveX** check box.
3. Click **OK** to close the **Extensions** dialog.

   When you're done testing the application, you may want to remove it from the **Extensions** dialog as well as from the **Extension Enabler** dialog.

Ignore an ActiveXVB class

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you are using the ActiveX/VB extension and you want to ignore a class, you must edit the `axext.ini` configuration file.

1. Open the `axext.ini` file, which is installed by default in the `<SilkTest installation directory>/extend` folder.
2. In the `[OmitClasses]` section, enter either the class names or class ids, separated by commas. For example:

   ```
   [OmitClasses]
   RawClassName=
   CLSID=00010001-0000-0000-0000-111111111111
   ```
3. Save and close `axext.ini`. The next time you open SilkTest, SilkTest ignores the class (or classes) you've listed.

Setting ActiveXVB extension options

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The ActiveX/VB options you can set are:

- `SetAdvSink` which sets the Advise Sink.
- `ShowAllWin` which controls whether SilkTest optimizes window hierarchy by skipping some windows.
- `MsgTimeout` which controls the timeout for messaging in the extension.

You can set the ActiveX/VB extension options by manually editing the `axext.ini` file. The settings go in the `[VBOptions]` section of `axext.ini` and are optional. You do not have to include them in your `axext.ini` if you want the default behavior.

SetAdvSink option

Default is TRUE. Setting the advise sink may cause certain applications to crash if they frequently destroy and recreate ActiveX/VB controls. Try setting this option to FALSE if your application under test crashes. Setting the option to FALSE disables the code in the ActiveX/VB extension that sets the advise sink.

ShowAllWin option

Default is FALSE. `ShowAllWin` lets you control whether or not SilkTest recognizes the full window hierarchy in ActiveX applications. The default FALSE setting causes the ActiveX extension to construct a

simplified window hierarchy that filters out windows perceived to be containers. Setting the option to TRUE causes the ActiveX extension to recognize the full window hierarchy. Try setting this option to TRUE if a control that you need to test is not recognized because it has the same position and size (in other words, the same rectangle) as another control that is recognized.

MsgTimeout option

Default is 1000. Setting the `MsgTimeout` option controls the number in milliseconds of the timeout used for messaging in the extension. 1000 milliseconds corresponds to 1 second. Try increasing the timeout to 2000 or 3000 (2 or 3 seconds) if you notice the following symptoms appearing at apparently random intervals:

- The VB/ActiveX extension recognizes certain controls as CustomWin rather than as 4Test or recorded classes.
- The application under test crashes while a test script is running.

Increasing the value of MsgTimeout may slow down SilkTest's performance when interacting with an application.

To edit the ActiveX/VB extension options

1. Close SilkTest and your application under test, if they are open.
2. Open axext.ini, located in the extend subdirectory of the directory where you installed SilkTest. If you are using a SilkTest Project, the applicable axext.ini file is still in the SilkTest install directory.
3. Go to the [VBOptions] section and change the value of the option. You may need to add a line containing the [VBOptions] section name, if it does not already exist.
4. Save and close the axext.ini file.
5. Restart SilkTest.

Setup for testing ActiveX controls or Java applets in the browser

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To test ActiveX controls in Microsoft's Internet Explorer, enable the extension for the version of the browser that you are using. When you enable the extension, be sure to check the **ActiveX** check box for the extension. This must be done manually whether you enable the extension manually or automatically.

To test Java applets in the browser, you must check the Java check box for the browser extension. In most cases, SilkTest detects the applet and automatically checks the check box.

For more information, see *Enabling extensions and Specifying your default browser*.

Rumba Support

Rumba is the world's premier Windows desktop terminal emulation solution. SilkTest provides built-in support for recording and replaying Rumba.

When testing with Rumba, please consider the following:

- The Rumba version must be compatible to the Silk4J version. Versions of Rumba prior to version 8.1 are not supported.
- All controls that surround the green screen in Rumba are using basic WPF functionality (or Win32).
- The supported Rumba desktop types are:
  - Mainframe Display
  - AS400 Display

For a complete list of the record and replay controls available for Rumba testing, see the *Rumba Class Reference*.

Enabling and Disabling Rumba

Rumba is the world's premier Windows desktop terminal emulation solution. Rumba provides connectivity solutions to mainframes, mid-range, UNIX, Linux, and HP servers.

Enabling Support

Before you can record and replay Rumba scripts, you need to enable support:

1. Install Rumba desktop client software version 8.1 or later.
2. Click **Start** > **Silk** > **SilkTest {version}** > **Administration Tools** > **Rumba plugin** > **Enable SilkTest Rumba plugin**.

Disabling Support

Click **Start** > **Silk** > **SilkTest {version}** > **Administration Tools** > **Rumba plugin** > **Disable SilkTest Rumba plugin**.

Locator Attributes for Identifying Rumba Controls

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

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

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

Rumba Class Reference

When you configure a Rumba application, SilkTest Classic automatically provides built-in support for testing standard Rumba controls.

Testing SAP Applications

Overview of SAP Support

SilkTest provides built-in support for testing SAP client/server applications based on the Windows-based GUI module.

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

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

SilkTest Agent Support

When you create a SilkTest SAP project, the Open Agent is assigned as the default Agent.

Note: You must set **Ctrl+Shift** as the shortcut key combination to use to pause recording. To change the default setting, click **OptionsRecorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.

Attributes for SAP Applications

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

Supported attributes for SAP include:

- *automationId*
- *caption*

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

Dynamically Invoking SAP Methods

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

Call dynamic methods on SAP objects with the `DynamicInvoke` method. To retrieve a list of supported dynamic methods for a SAP control, use the `GetDynamicMethodList()` method.

Retrieve dynamic properties with the `GetProperty` method and set dynamic properties with the `SetProperty()` method. To retrieve a list of supported dynamic properties for a SAP control, use the `GetPropertyList()` method.

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

Supported Methods and Properties

The following methods and properties can be called:

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

Supported Parameter Types

The following parameter types are supported:

All built-in SilkTest types	SilkTest types include primitive types, such as boolean, int, and string, lists, and other types, such as Point and Rect.
UI controls	UI controls can be passed or returned as WINDOW.

Returned Values

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

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

Example

A custom calculator control has a `Reset` method and an `Add` method, which performs an addition of two numbers. You can use the following code to call the methods directly from your tests:

```
customControl.Invoke("Reset")
REAL sum = customControl.DynamicInvoke("Add",{1,2})
```

SAP Class Reference

When you configure an SAP application, SilkTest Classic automatically provides built-in support for testing standard SAP controls.

Testing Windows-Based Applications

Overview of Windows-Based Application Support

SilkTest provides built-in support for testing Microsoft Windows API-based applications. Several objects exist in Microsoft applications that SilkTest can better recognize if you enable Accessibility. For example, without enabling Accessibility SilkTest records only basic information about the menu bar in Microsoft Word and the tabs that appear in Internet Explorer 7.0. However, with Accessibility enabled, SilkTest fully recognizes those objects. You can also improve SilkTest object recognition by defining a new window, if necessary.

You can test Windows API-based applications using the SilkTest Classic or Open Agent.

Sample Applications

SilkTest provides sample Windows API-based test applications that you can use with the SilkTest Open and Classic Agents. You must download the sample applications from *http://techpubs.borland.com/silk_gauntlet/SilkTest/*. After you have installed the sample applications, choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Sample Applications** > **Win32** to select the sample application that is right for your environment.

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

Object Recognition

Windows API-based applications support hierarchical object recognition and dynamic object recognition. You can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements.

When you record a testcase with the Open Agent, SilkTest creates locator keywords in an INC file to create scripts that use dynamic object recognition and window declarations.

Existing testcases that use dynamic object recognition without locator keywords in an INC file will continue to be supported. You can replay these tests, but you cannot record new tests with dynamic object recognition without locator keywords in an INC file.

To test Windows API-based applications using hierarchical object recognition, record a test for the application that you want to test. Then, replay the tests at your convenience.

Supported Controls

For a complete list of the record and replay controls available for Windows-based testing for each Agent type, view the WIN32.inc and winclass.inc file. To access the `WIN32.inc` file, which is used with the Open Agent, navigate to the `<SilkTest directory>\extend\WIN32` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\extend\WIN32\WIN32.inc`. To access the `winclass.inc` file, which is used with the Classic Agent, navigate to the `<SilkTest directory>\` directory. By default, this file is located in `C:\Program Files\Silk\SilkTest\winclass.inc`.

Attributes for Windows API-based ClientServer Applications

The attributes that SilkTest supports for Windows API-based client/server applications include:

- `caption` (supports wildcards ? and *)
- `windowid`
- `priorlabel` (For controls that do not have a caption, `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.

Improving SilkTest object recognition with Accessibility

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

There are several objects in Internet Explorer and Microsoft applications that SilkTest can better recognize if you enable accessibility. For example, without enabling accessibility SilkTest records only basic information about the menu bar in Microsoft Word and the tabs that appear in Internet Explorer 7.0. However, with accessibility enabled, SilkTest fully recognizes those objects.

Accessibility is not available for Java applications or applets.

Comparison of what SilkTest records

Without Accessibility enabled on Microsoft Excel, SilkTest records the following if the mouse points to the File command on the main toolbar:

```
[+] CustomWin MenuBar
[+] multitag "[MsoCommandBar]Menu Bar"
[ ] "[MsoCommandBar]$0[1]"
[+] CustomWin TypeAQuestionForHelp
[+] multitag "[RichEdit20W]Type a question for help"
[ ] "[RichEdit20W]$16075636"
[+] CustomWin Standard
[+] multitag "[MsoCommandBar]Standard"
[ ] "[MsoCommandBar]$0[2]"
...
```

However, if you record the same testcase with accessibility enabled, SilkTest is able to record the following:

```
[+] AccObject WorksheetMenuBar
    [+] multitag "Worksheet Menu Bar"
        [ ] "$window"
[+] AccObject WorksheetMenuBar2
    [+] multitag "Worksheet Menu Bar[2]"
     [ ] "$menu bar[2]"
    [+] AccMenuItem File
        [+] multitag "File"
            [ ] "$menu item[1]"
    [+] AccMenuItem Edit
        [+] multitag "Edit"
         [ ] "$menu item[2]"
    [+] AccMenuItem View
        [+] multitag "View"
            [ ] "$menu item[3]"
    [+] AccMenuItem Insert
        [+] multitag "Insert"
            [ ] "$menu item[4]"
    [+] AccMenuItem Format
        [+] multitag "Format"
            [ ] "$menu item[5]"
…
```

With accessibility enabled SilkTest is able to record more than simple details about the File menu command.

SilkTest stores the information about accessibility classes in the accex.inc file which is installed by default in the `<SilkTest installation directory>/Extend or <name of project>\extend` directory. The `accex.inc` file comes preloaded with several classes, including the MsoCommandBar, the class of the Microsoft Office menu bar.

Improving SilkTest Window Declarations

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The current methodology for identifying window declarations in Microsoft Windows-based applications during a recording session is usually successful. However, some applications may require an alternate approach of obtaining their declarations because their window objects are invisible to the SilkTest Recorder. You can try any of the following:

- Turning on Accessibility - use this if during a session started with the Recorder, SilkTest is unable to recognize objects within a Microsoft Windows-based application. This functionality is available only for projects or scripts that use the SilkTest Classic Agent.
- Defining a new window - use this if turning on Accessibility does not help SilkTest to recognize the objects. This functionality is available only for projects or scripts that use the SilkTest Classic Agent.
- Creating a testcase that uses dynamic object recognition - use this to create testcases that use XPath queries to find and identify objects. Dynamic object recognition uses a **Find** or **FindAll** method to identify an object in a testcase. This functionality is available only for projects or scripts that use the SilkTest Open Agent.

Suppressing Controls for Certain Classes

You can suppress the controls for certain classes for .NET, Java SWT, and Windows API-based applications. For example, you might want to ignore container classes to streamline your testcases. Ignoring these unnecessary classes simplifies the object hierarchy and shortens the length of the lines of code in your test scripts and functions. Container classes or ‘frames’ are common in GUI development, but may not be necessary for testing.

Common classes that are suppressed include:

.NET	Group
Java SWT	org.eclipse.swt.widgets.Composite
	org.eclipse.swt.widgets.Group
Windows API-based applications	Group

To suppress the controls for .NET applications, you must use the Classic Agent. To suppress the controls for Java SWT and Windows API-based applications, you must use the Open Agent.

1. Click **Options** > **Class Map**.
   The **Class Map** dialog opens.
2. In the Custom class field, type the name of the class that you want suppress.

   The class name depends on the technology and the extension that you are using. For Windows API-based applications, use the Windows API-based class names. For Java SWT applications, use the fully qualified Java class name.

   To ignore the **SWT_Group** in a Windows API-based application, type `SWT_Group`. For Java SWT applications, type `org.eclipse.swt.widgets.Group`.
3. In the **Standard class** list, select **Ignore**.
4. Click **Add**. The custom class and the standard class display at the top of the dialog.

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

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)

Classic Agent	Open Agent
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>
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>
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

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

Classic Agent	Open Agent
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*.

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

The Classic Agent and the Open Agent differ slightly in the function parameters 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.

For some parameters, the Open Agent uses a hard-coded default value internally. If one of these parameters is set in a 4Test script, the Open Agent ignores the value and uses the value listed here.

Function	Parameter	Classic Agent Value	Open Agent Value
AnyWin::PressKeys/ ReleaseKeys	nDelay	Any number.	0
AnyWin::PressKeys/ ReleaseKeys	sKeys	More than one key is supported.	Only one key is supported. The first key is used and the remaining keys are ignored. For example `MainWin.PressKeys( "<Shift><Left>")` will only press the **Shift** key. To press both keys, specify `MainWin.PressKeys( "<Shift>") MainWin.PressKeys( "<Left >")`.
AnyWin::TypeKeys	sEvents	Keystrokes to type or mouse buttons to press.	The Open Agent supports keystrokes only.
AnyWin::GetChildren	bInvisible	TRUE or FALSE.	FALSE.
AnyWin::GetChildren	bNoTopLevel	TRUE or FALSE.	FALSE.
TextField::GetFontName	iLine	The Classic Agent recognizes this parameter.	The Open Agent ignores this parameter.
AnyWin::GetCaption	bNoStaticText	TRUE or FALSE.	FALSE.
AnyWin::GetCaption, Control::GetPriorStatic	bRawMode	TRUE or FALSE.	FALSE. However, the returned strings include trailing and leading spaces, but ellipses, accelerators, and hot keys are removed.
PageList::GetContents/ GetPageName	bRawMode	TRUE or FALSE.	FALSE. However, the returned strings include trailing and leading spaces, ellipses, and hot keys but accelerators are removed.

Function	Parameter	Classic Agent Value	Open Agent Value
AnyWin::Click/ DoubleClick/ MoveMouse/ MultiClick/ PressMouse/ ReleaseMouse, PushButton::Click	bRawEvent	The Classic Agent recognizes this parameter.	The Open Agent ignores this value.

Configuring Standard Applications

A standard application is an application that does not use a Web browser, such as a Windows application or Java SWT application.

Configure the application that you want to test to set up the environment that SilkTest will create each time you record or replay a test case.

1. Start the application that you want to test.
2. Click **Configure Application** on the basic workflow bar.

   If you do not see **Configure Application** on the workflow bar, ensure that the default agent is set to the Open Agent.

   The **New Test Frame** dialog box opens.
3. Double-click Standard Test Configuration.
   The **Standard Configuration** page opens.
4. Click the configuration that you want to test.

   For example, to test the sample Java SWT application, click `javaw.exe` with the Swt Test Application caption.
5. To specify a command-line pattern that SilkTest uses to configure the application, check the **Optional command line pattern** check box.

   SilkTest automatically fills in the command line that matches the application that you selected. SilkTest will only enable the processes that match the module pattern and the command-line pattern. You can modify the command-line pattern to meet your needs. It is case insensitive and allows an asterisk (*) as a wild card, which matches any text of any length, and a question mark (?), which matches one character.

   Using the command line is especially useful for Java applications because most Java programs run by using `javaw.exe` from the Java installation directory configured with a classpath and a JAR file. This means that when you configure the SWT sample application, the pattern, `*\javaw.exe` is used, which matches any Java process. Using the command line ensures that only the application that matches the command line is used.
6. Click **Next**.
   The **Base State Configuration** page opens. By default, SilkTest lists the caption of the main window of the application as the locator for the base state. An application's base state is the known, stable state that you expect the application to be in before each test begins execution, and the state the application can be returned to after each test has ended execution. This state may be the state of an application when it is first started.
7. *Optional:* To select a different locator for the base state, click **Pick Locator** and perform the following steps:
   a) Position the mouse over the object in the application that you want to use as the locator.
   b) Press **Ctrl+Alt** when the object that you want to use displays in the text box.

Note: For SAP applications, you must set **Ctrl+Alt** as the shortcut key combination to use. To change the default setting, click **Options** > **Recorder** and then check the **OPT_ALTERNATE_RECORD_BREAK** check box.

8. Click Finish.
   The **Choose name and folder of the new frame file** page opens. SilkTest configures the recovery system and names the corresponding file `frame.inc` by default.
9. Navigate to the location in which you want to save the frame file.
10. In the **File name** text box, type the name for the frame file that contains the default base state and recovery system. Then, click **Save**.

    SilkTest automatically creates a base state for the application. When you configure an application, SilkTest adds an include file based on the technology or browser type that you enable to the **Use files location** in the **Runtime Options** dialog box.

    SilkTest opens the include file.
11. Record the test case whenever you are ready.

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.

Distributed Testing - Testing on Multiple Machines

Configuring Your Test Environment

Overview of Configuring Tasks

This topic contains information about configuration tasks.

PC-class platforms

You have to explicitly assign a unique network name to remote Agents so that SilkTest can identify the Agent when your testcase connects to that machine.

TCP/IP

On PCs. Windows machines generally come with TCP/IP. SilkTest on Microsoft Windows can use any TCP/IP software package that supports the Windows Sockets Interface Standard, Version 1.1, (WINSOCK), and supplies WINSOCK.DLL.

LAN Manager or Windows for Workgroups

For LAN Manager networks or Windows for Workgroups, you might have to increase the SESSIONS value (the default is 6) to a higher value. This variable is defined in the `protocol.ini` file, which is typically located in your Windows directory. You should also increase the NCBS value in `protocol.ini` to twice the SESSIONS value.

The LAN Manager network environment and Windows for Workgroups have the ability to use more than one protocol driver at a time. NetBEUI is the protocol driver frequently used by LAN Manager. In order for SilkTest and the Agent to run, the NetBEUI protocol must be the first protocol loaded. The LANABASE option under the [NETBEUI_XIF] section of `protocol.ini` must be set to 0 (zero). If additional protocols are loaded, they must have a sequentially higher LANABASE setting. For example, if you are running both NetBEUI and TCP/IP, the LANABASE setting for NetBEUI is (as always) 0 (zero), and the value for TCP/IP is1 (one).

NetBIOS on PCs

Under Windows, you must install NetBEUI with NetBIOS. In the Network control panel, you must have NetBEUI set as the default protocol.

On Windows, NetBIOS is started automatically.

You have to explicitly assign a unique network name to remote Agents so that SilkTest can identify the Agent when your testcase issues a Connect function for that machine. This step is not necessary for Agents using TCP/IP because SilkTest automatically uses the workstation's TCP/IP name.

The name must be from 1 to 16 alphanumeric characters long and must not be the standard name you use for your machine itself or the name of any other distributed Agent. On some systems, using the same name can cause a system crash. A safe alternative is to derive the Agent name from the machine name. For example, if a machine is called Rome, call the Agent Rome_QAP.

Your NetBIOS adapter may be configured as any host adapter number. In the past, SilkTest could only be configured as adapter 0, but this is no longer the case. Check with your network administrator if you are not sure how to do this or need to change your configuration.

ClientServer Testing Configurations

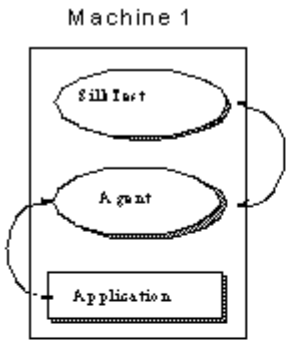
The processes that participate in a client/server testing scenario are logically associated with three different computers:

1. System A runs SilkTest, which processes test scripts and sends application commands to the Agent.
2. System B runs the client application and the Agent, which submits the application commands to the client application.
3. System C runs the server software, which reacts to requests submitted by the client application.

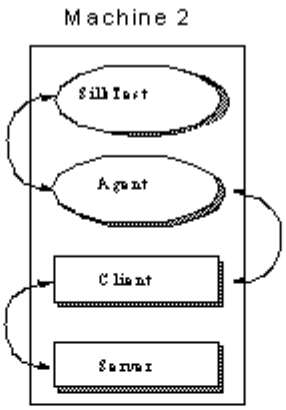
The following sections describe different hardware/software configurations that can support SilkTest testing.

Configuration 1

Machine 1 shows the software configuration you would have when testing a stand-alone application. Machine 2 shows SilkTest and a client/server application with all of your software running on one machine. This configuration allows you to do all types of functional testing other than testing the behavior of the connection between a client and a remote server.



Testing a stand-alone application with SilkTest



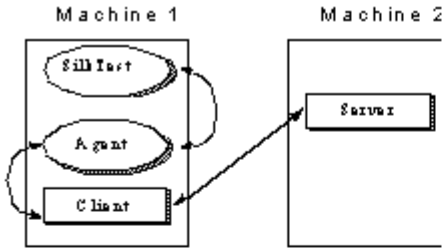
Testing a client/server application with software all in one machine

During your initial test development phase, you can reduce your hardware needs by making two (and possibly all) of these systems the same. If you write tests for an application running on the same system as SilkTest, you can implement the tests without consideration of any of the issues of remote testing. You can then expand your testing program incrementally to take your testing into each new phase.

Configuration 2

A testing configuration in which the client application runs on the same machine as SilkTest and the server application runs on a separate machine.

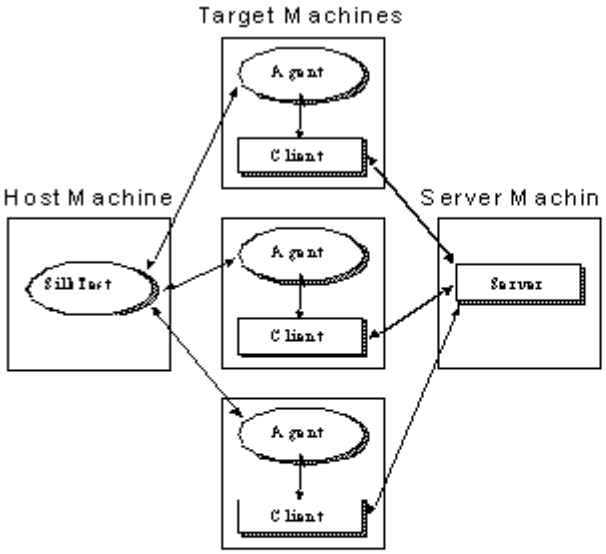
Note: In this configuration, as with Machine 2 in Configuration 1, there is no communication between SilkTest and the server. This means that you must manage the work of starting and initializing the server database manually. For some kinds of testing this is appropriate.



This configuration lets you test the remote client-to-server connection and is appropriate for many stress tests. It allows you to do volume load testing from the point of view of the client application, but not the server.

Configuration 3

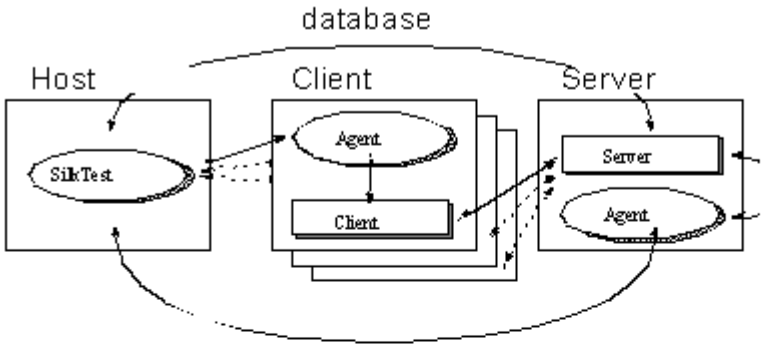
Multiple copies of the client application running on separate machines, with SilkTest driving the client application by means of the Agent process on each client machine, and the client application driving the server application. This is just the multi-client version of the previous configuration. You could run a fourth instance of the client application on the SilkTest machine. The actual number of client machines used is your choice.



This configuration is appropriate for load testing and configuration testing if you have no need to automatically manipulate the server. You must have at least two clients to test concurrency and mutual-exclusion functionality.

Figure 4

Once you are running SilkTest, it makes sense to have your script initialize your server automatically. Configuration 4 uses the same hardware configuration as Configuration 3, but SilkTest is also driving the server directly. This figure shows SilkTest using an Agent on the server machine to drive the server's GUI (the lower connecting arrow); this approach can be used to start the server's database and sometimes can be used to initialize it to a base state. The upper arrow shows SilkTest using SQL commands to directly manipulate the server database; use this approach when using the Agent is not sufficient. After starting the database with the Agent, use SQL commands to initialize it to a base state. The SQL commands are submitted by means of SilkTest's database functions, which do not require the services of the Agent.



Configuration 4 is the most complete testing configuration. It requires the database tester. You can use it for all types of SilkTest testing, including volume load testing of the server, peak load testing, and performance testing.

The special features that allow SilkTest to provide rigorous testing for client/ server applications are the following:

- Automatic control of multiple applications.
- Multithreading for automatic control of concurrent applications.
- Reporting results by thread ID.
- Testing across networks using a variety of protocols.

The added value that the database tester provides for the client/server environment is direct database access from the test script.

Networking Protocols Used by SilkTest

This functionality is available only for projects or scripts that use the Classic Agent. The Open Agent uses exclusively the TCP/IP protocol.

SilkTest runs on many platforms, but only three different protocols are used. This means that a SilkTest script on one platform can drive the Agent on a target platform, as long as both the host and Agent platforms are running an appropriate protocol for the platform and both are running the same protocol, regardless of the protocols used by the applications under test. The table lists the protocols available for each platform.

Platform	TCP/IP	NetBIOS/ NetBEUI
Windows	•	•
AIX	•	
IRIX	•	

Suppose you are running SilkTest under Windows and you are testing an application that requires TCP/IP communications in order to communicate with a server on a Sun Sparc station. SilkTest's Windows

machine can run NetBIOS for the host and the application's Windows machine must then run NetBIOS for the Agent as well as TCP/IP for the application under test. Running NetBIOS will have no impact on your TCP/ IP connections but will allow SilkTest to communicate with the Agent. Alternatively, since the application is already running TCP/IP, you can choose to use TCP/IP for SilkTest and its Agents as well.

There is no limit on the protocol or API that an application under test may use. Just make sure that the protocol required by SilkTest and the protocol required by your application are running at the same time.

Single Local Applications

In a single-application test environment, if the application is local, you do not have to determine an Agent name or issue a connection command. When you start an Agent on the local machine, SilkTest automatically connects to it and directs all Agent commands to it.

Single Remote Applications

In a single-application test environment, if the application is remote, specify the Agent name in the **Runtime Options** dialog box. This causes SilkTest to automatically connect to that machine and to direct all Agent commands to that machine. This contrasts with the multi-application case, in which you explicitly connect to the target machines and explicitly specify which machines are to receive which sections of code.

Remote Applications

When you have one or more remote Agents in your testing network, you enable networking by specifying the network type. For projects or scripts that use the Classic Agent, if you are not using TCP/IP, you have to assign to each Agent the unique name that your scripts use to direct test operations to the associated application. For additional information, see *Enabling Networking and Assigning a Network Name to the Agent*.

You can use SilkTest to test two applications on the same target from one host machine.

Multiple Remote Applications

When you enable networking by selecting the networking type in the **Runtime Options** dialog box on the host, do not set the **Agent Name** text box to an Agent name if you have multiple remote Agents. This field only accepts a single Agent name and using it prevents you from handling all your client machines the same way.

If you specify one Agent name from your set of Agents, then you cannot issue a `Connect` call to that Agent and thus do not receive the machine handle that the `Connect` function returns. Since you have to issue some `Connect` calls, be consistent and avoid writing exception code to handle a machine that is automatically connected.

For projects or scripts that use the Classic Agent, you can specify multiple Agents from within your script file by adding the following command line to the Agent:

```
agent –p portNumber
```

Configuring a Network of Computers

To configure a network of computers so that they can run SilkTest and its Agents, you must perform the following steps:

1. Install, or have already running, networking protocols supported by SilkTest.
2. Install SilkTest on the host machine and the Agent software on all target machines.
3. Establish connectability between host and Agents.

   This may be automatic or may require some setup, depending on the circumstances.
4. Enable networking on any target machines.

Use the Agent window, as described in *Enabling Networking and Assigning a Network Name to the Agent*.

5. Enable networking on the host machine.

   Use the **Runtime Options** dialog box. Details may vary, depending on your configuration.

6. Gather the information that your test scripts need when making explicit connections.

   For example, you can edit the Agent names into a list definition and have your test plan pass the list variable name as an argument for testcases controlled by that plan. The testcases then pass each Agent name to a `Connect` or `SetUpMachine` function and that function makes the explicit host-to-Agent connection.

Configuration details are specific to the different protocols and operating systems you are using. In general, set up your Agents and make all adjustments to the `partner.ini` file or environment variables before enabling networking on the host machine.

Enabling Networking and Assigning the SilkTest Classic Agent Name and Port

This describes the process for enabling the SilkTest Classic Agent on Windows. You only need to execute this procedure the first time you run the Agent.

1. Start the SilkTest Classic Agent.
2. Right-click the **Agent** icon and choose **Network**.
   The **Agent Network** dialog box displays.
3. Select the network type from the **Network** list box.
4. For TCP/ IP, the default port number displays in the **Port number** text box. Typically, you accept the default.
5. For NetBIOS, type the Agent name in the **Agent name** text box in the format `hostname:#` where hostname is a unique host name on your network and # is the host adapter number (for example, dwc: 3).

   Your NetBIOS adapter may be configured as any host adapter number. In the past, SilkTest could only be configured as adapter 0, but this is no longer the case. Check with your network administrator if you are not sure how to do this or need to change your configuration.
6. Click **OK**.

Enabling Networking on NetBIOS Host

This functionality is available only for projects or scripts that use the Classic Agent.

Once the protocol has been selected for all the Agents and they are named, you can enable networking on the host. Do this by choosing **Options** > **Runtime** and selecting the NetBIOS network type. Then fill in the Agent name if you have a single-remote-application configuration.

Your NetBIOS adapter may be configured as any host adapter number. In the past, SilkTest could only be configured as adapter 0, but this is no longer the case. Check with your network administrator if you are not sure how to do this or need to change your configuration.

Enabling Networking on an Agent

This functionality is available only for projects or scripts that use the Classic Agent.

You assign the selected name to the Agent when you enable networking for each Agent PC.

For each Agent, the Agent name that you specify in the `Connect` function is the name of that Agent host, stored in the network host database. You can find the host name in the name given to the Agent icon. The name takes the form `Agent [TCP/IP <Host Name> <Port Number>]`.

For Windows, move the mouse pointer over the Agent icon and wait two seconds; the icon name displays.

You must enable networking for each Agent PC by selecting the protocol type to be used in the Agent window. When you select TCP/IP as the protocol, the port number field is displayed with the default TCP/IP port number. When you click **OK**, the selection is accepted if the default port is available.

Enabling Networking on a Remote Host

Once the protocol has been picked for any PC Agents and the port settings are consistent, you can enable networking on the host.

Do this by choosing **Options** > **Runtime** and setting the port number and/or Agent name. You can skip this step if you do not have to change the default port number and you are not specifying an Agent name for a single-remote-application configuration.

For additional information, see *Configuring your Test Environment*.

Configuring SilkTest Classic Open Agent Port Numbers

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. 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.

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.

After changing the port number, restart the Open Agent, SilkTest Classic, Silk4J, SilkTest Recorder, and the application that you want to test.

Running Testcases in Parallel

Overview of SilkTests Parallel Processing Features

A concurrent, or multithreaded, script is one in which multiple statements can execute in parallel. Concurrency allows you to more effectively test distributed systems, by permitting multiple client applications to submit requests to a server simultaneously.

The 4Test language fully supports the development of concurrent scripts which enables a script to:

- Create and coordinate multiple concurrent threads.
- Protect access to variables, which are global to all threads.
- Synchronize threads with semaphores.
- Protect critical sections of code for atomic operations.
- Recover from errors in the event of script deadlock.

Concurrency

For SilkTest, concurrent processing means that Agents on a specified set of machines drive the associated applications simultaneously. To accomplish this, the host machine interleaves execution of the sets of code assigned to each machine. This means that when you are executing identical tests on several machines, each machine can be in the process of executing the same operation. For example, select the Edit.FindChange menu item.

At the end of a set of concurrent operations, you will frequently want to synchronize the machines so that you know that all are ready and waiting before you submit the next operation. You can do this easily with 4Test.

There are several reasons for executing testcases concurrently:

- You want to save testing time by running your functional tests for all the different platforms at the same time and by logging the results centrally, on the host machine.
- You are testing cross-network operations.
- You need to place a multi-user load on the server.
- You are testing the application's handling of concurrent access to the same database record on the server.

To accomplish testing concurrent database accesses, you simply set all the machines to be ready to make the access and then you synchronize. When all the machines are ready, you execute the operation that commits the access operation—for example, clicking the **OK** button. Consider the following example:

```
// [A] Execute 6 operations on all machines concurrently
for each sMachine in lsMachine
  spawn
    SixOpsFunction (sMachine)
rendezvous                        // Synchronize

// [B] Do one operation on each machine
for each sMachine in lsMachine
  spawn
    [sMachine]MessageBox.OK.Click () // One operation
rendezvous                              // Synchronize
```

In code fragment [A], the six operations defined by the function `SixOpsFunction` are executed simultaneously on all machines in a previously defined list of Agent names. After the parallel operation, the script waits for all the machines to complete; on completion, they will present a message box, unless the application fails. In code fragment [B], the message box is dismissed. By putting the message dismissal operation into its own parallel statement block instead of adding it to the `SixOpsFunction`, you are able to synchronize and all machines click at almost the same instant.

In order to specify that a set of machines should execute concurrently, you use a 4Test command that starts concurrent threads. In the fragments above, the spawn statement starts a thread for each machine. For additional information, see *Threads and Concurrent Programming*.

Global Variables

Suppose the code for each machine is counting instances of some event. You want a single count for the whole test and so each machine adds its count to a global variable. When you are executing the code for all your machines in parallel, two instances of the statement iGlobal = iGlobal + iCount could be executing in parallel. Since the instructions that implement this statement would then be interleaved, you could get erroneous results. To prevent this problem, you can declare a variable shareable. When you do so, you can use the access statement to gain exclusive access to the shared variable for the duration of the block of code following the access statement. Make variables shareable whenever the potential for conflict exists.

Recovering Multiple Tests

There are three major categories of operations that an Agent executes on a target machine:

- Setup operations that bring the application to the state from which the next test will start.
- Testing operations that exercise a portion of the application and verify that it executed correctly.
- Cleanup operations that handle the normal completion of a test plus the case where the test failed and the application is left in an indeterminate state. In either case, the cleanup operations return the application to a known base state.

When there are multiple machines being tested and more than one application, the Agent on each machine must execute the correct operations to establish the appropriate state, regardless of the current state of the application.

Remote Recording

Once you establish a connection to a target machine, any action you initiate on the host machine, which is the machine running SilkTest, is executed on the target machine.

With the Classic Agent, one Agent process can run locally on the host machine, but in a networked environment, the host machine can connect to any number of remote Agents simultaneously or sequentially. You can record and replay tests remotely using the Classic Agent. If you initiate a Record/Testcase command on the host machine, you record the interactions of the user manipulating the application under test on the target machine. In order to use the Record menu's remote recording operations, you must place the target machine's name into the **Runtime Options** dialog box. Choose **Options** > **Runtime**.

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

Threads and Concurrent Programming

SilkTest can run testcases in parallel on more than one machine. To run testcases in parallel, you can use parallel threads within `main( )` or in a function called by `main( )`. If you attempt to run testcases in parallel on the same machine, you will generate a runtime error.

A more elegant alternative to parallel threads is to use a multitestcase function, which provides a robust multi-machine recovery system. For additional information on multitestcase code templates, see *Using the Client/Server Template* and *Using the Parallel Template*.

In the 4Test environment, a thread is a mechanism for interleaving the execution of blocks of client code assigned to different Agents so that one script can drive multiple client applications simultaneously. A thread is part of the script that starts it, not a separate script. Each thread has its own call stack and data stack. However, all the threads that a script spawns share access to the same global variables, function arguments, and data types. A file that one thread opens is accessible to any thread in that script.

While the creation of a thread carries no requirement that you use it to submit operations to a client application, the typical reason for creating a multithread script is so that each thread can drive test functions for one client, which allows multiple client application operations to execute in parallel.

When a script connects to a machine, any thread in that script is also connected to the machine. Therefore, you must direct the testing operations in a thread to a particular Agent machine. Threads interleave at the machine instruction level; therefore, no single 4Test statement is atomic with respect to a statement in another thread.

Driving Multiple Machines

When you want to run tests on multiple machines simultaneously, you connect to all the machines and then you direct specific test operations to particular machines. This enables you to drive different applications concurrently. For example, you can test the intercommunication capabilities of two different applications or you can drive both a client application and its server.

To do this, at the beginning of a test script you issue for each machine an explicit connection command. This can be either `Connect`(*agent_name*) or `SetMachine`(*agent_name*). This connection lasts for the duration of the script unless you issue a `Disconnect`(*agent_name*) command. In the body of the script you can specify that a particular portion of code is to be executed on a particular machine. The `SetMachine`(*agent_name*) command specifies that the following statements are directed to that Agent. You can specify that just one statement is directed to a particular Agent by using the bracket form of the machine handle operator. For example `["Client_A"]SYS_SetDir ("c:\mydir")`.

Since 4Test allows you to pass variables to these functions, you can write a block of code that sends the same operations to a particular set of target machines and you can pass the `SetMachine` function in that

block of code a variable initialized from a list that specifies the machines in that set. Thus, specifying which machines receive which operations is very simple.

Protecting Access to Global Variables

When a new thread is spawned, 4Test creates a new copy of all local variables and function arguments for it to use. However, all threads have equal access to global variables. To avoid a situation in which multiple threads modify a variable simultaneously, you must declare the variable as shareable. A shareable variable is available to only one thread at a time.

Instances where threads modify variables simultaneously generate unpredictable results. Errors of this kind are difficult to detect. Make variables shareable wherever the potential for conflict exists.

A declaration for a shareable variable has the following form:

```
[scope] share data-type name [= expr] {, name [= expr]}
```

- *scope* can be either public or private. If omitted, the default is public.
- *data-type* is a standard or user-defined data type.
- *name* is the identifier that refers to the shareable variable.
- *expr* is an expression that evaluates to the initial value you want to give the variable. The value must have the same type you gave the variable. If you try to use a variable before its value is set, 4Test raises an exception.

At any point in the execution of a script, a shared variable can only be accessed from within the block of code that has explicitly been granted access to it. You request access to shareable variables by using the access statement.

An access statement has the following form:

```
access name1, name2, ...
  statement
```

where *name1*, *name2*, ... is a list of identifiers of optional length, each of which refers to a shareable variable and `statement` is the statement to be executed when access to the variables can be granted.

If no other thread currently has access to any of the shareable variables listed, 4Test executes the specified statement. Otherwise, 4Test blocks the thread where the `access` statement occurs until access can be granted to all the shareable variables listed. At that point, 4Test blocks competing threads and executes the blocked thread.

Example

```
share INTEGER iTestNum = 0
public share STRING asWeekDay [7]
share ANYTYPE aWhoKnows

void IncrementTestNum ()
  access iTestNum
    iTestNum = iTestNum + 1
```

Synchronizing Threads with Semaphores

Use semaphores to mutually exclude competing threads or control access to a resource. A semaphore is a built-in 4Test data type that can only be assigned a value once. The value must be an integer greater than zero. Once it is set, your code can get the semaphore's value, but cannot set it.

Example

The following code example shows legal and illegal manipulations of a variable of type SEMAPHORE:

```
SEMAPHORE semA = 10                          // Legal
semA = 20                                    // Illegal -
```

```
existing semaphore
                                          // cannot be
reinitialized
if (semA == [SEMAPHORE]2)...              // Legal - note the
typecast
Print ("SemA has {semA} resources left.")  // Legal
SEMAPHORE semB = 0                        // Illegal - must be
greater than 0
```

To compare an integer to a semaphore variable, you must typecast from integer to semaphore using `[SEMAPHORE]`.

Note: You cannot cast a semaphore to an integer.

To use a semaphore, you first declare and initialize a variable of type SEMAPHORE. Thereafter, 4Test controls the value of the semaphore variable. You can acquire the semaphore if it has a value greater than zero. When you have completed your semaphore-protected work, you release the semaphore. The `Acquire` function decrements the value of the semaphore by one and the `Release` function increments it by one. Thus, if you initialize the semaphore to 5, five threads can simultaneously execute semaphore-protected operations while a sixth thread has to wait until one of the five invokes the `Release` function for that semaphore.

The `Acquire` function either blocks the calling thread because the specified semaphore is zero, or "acquires" the semaphore by decrementing its value by one. `Release` checks for any threads blocked by the specified semaphore and unblocks the first blocked thread in the list. If no thread is blocked, `Release` "releases" the semaphore by incrementing its value by one so that the next invocation of `Acquire` succeeds, which means it does not block.

A call to `Acquire` has the following form:

```
void Acquire(SEMAPHORE semA)
```

Where `semA` s the semaphore variable to acquire.

A call to `Release` has the following form:

```
void Release(SEMAPHORE semA)
```

Where `semA` s the semaphore variable to release.

If more than one thread was suspended by a call to `Acquire`, the threads are released in the order in which they were suspended.

A semaphore that is assigned an initial value of 1 is called a binary semaphore, because it can only take on the values 0 or 1. A semaphore that is assigned an initial value of greater than one is called a counting semaphore because it is used to count a number of protected resources.

Example: Application only supports three simultaneous users

Suppose you are running a distributed test on eight machines using eight 4Test threads. Assume that the application you are testing accesses a database, but can support only three simultaneous users. The following code uses a semaphore to handle this situation:

```
SEMAPHORE DBUsers = 3
...
Acquire (DBUsers)
  access database
Release (DBUsers)
```

The declaration of the semaphore is global; each thread contains the code to acquire and release the semaphore. The initial value of three ensures that no more than three threads will ever be executing the database access code simultaneously.

Testing In Parallel but Not Synchronously

This topic illustrates a method for running test functions in parallel on multiple clients, but with different tests running on each client. This provides a realistic multi-user load as opposed to a load in which all clients perform the same operations at roughly the same time.

Example

This example suggests a method by which each client, operating in a separate thread, executes a test that is assigned by a random number. The `RandSeed` function is called first so that the random number sequence is the same for each iteration of this multi-user test scenario. This enables you to subsequently repeat the test with the same conditions.

The example reads a list of client machines from a file, `clients.txt`, and receives the test count as in input argument. These external variables make the example scalable as to the number of machines being tested and the number of tests to be run on each. The number of different testcases is twelve in this example, but could be changed by modifying the `SelectTest` function and adding further test functions. For each machine in the client machine list, the example spawns a thread in which the specified client executes a randomly selected test, repeating for the specified number of tests.

Note: You can execute this test as it is written because it sets its own application states. However, when you use multi-application support, this is automatic. And if you want to use this approach to drive different applications or to initialize a server before starting the testing, you must add multi-application support.

```
testcase ParallelRandomLoadTest (INTEGER iTestCount)
  LIST OF STRING lsClients
  RandSeed (3)

  // list of client names
  ListRead (lsClients, "clients.txt")

  STRING sClientName

  for each sClientName in lsClients
    spawn
      // Connect to client, which becomes current machine
      Connect (sClientName)
      SetAppState ("MyAppState")          // Initialize
application
      TestClient (iTestCount)
      Disconnect (sClientName)
    rendezvous

  TestClient (INTEGER iTestCount)
    for i = 1 to iTestCount
      SelectTest ()

  SelectTest ()
    INTEGER i = RandInt (1, 12)

    // This syntax invokes Test1 to Test12, based on i
    @("Test{i}") ()
```

```
// Define the actual test functions
Test1 ()
  // Do the test . . .

Test2 ()
  // Do the test . . .
. . .
Test12 ()
  // Do the test . . .
```

Statement Types

Parallel Processing Statements

You create and manage multiple threads using combinations of the 4Test statements parallel, spawn, rendezvous, and critical.

In 4Test, all running threads, which are those not blocked, have the same priority with respect to one another. 4Test executes one instruction for a thread, then passes control to the next thread. The first thread called is the first run, and so on.

All threads run to completion unless they are deadlocked. 4Test detects script deadlock and raises an exception.

Note: The 4Test exit statement terminates all threads immediately when it is executed by one thread.

Using Parallel Statements

A parallel statement spawns a statement for each machine specified and blocks the calling thread until the threads it spawns have all completed. It condenses the actions of spawn and rendezvous and can make code more readable.

The parallel statement executes a single statement for each thread. Thus if you want to run complete tests in parallel threads, use the invocation of a test function, which may execute many statements, with the parallel statement, or use a block of statements with spawn and rendezvous.

To use the parallel statement, you must specify the machines for which threads are to be started. You can follow the parallel keyword with a list of statements, each of which specifies a different Agent name. For example:

```
parallel
  DoSomething ("Client1")
  DoSomething ("Client2")
```

The `DoSomething` function then typically issues a `SetMachine(sMachine)` call to direct its machine operations to the proper Agent.

Using a Spawn Statement

A spawn statement begins execution of the specified statement or block of statements in a new thread. Since the purpose of spawn is to initiate concurrent test operations on multiple machines, the structure of a block of spawned code is typically:

- A `SetMachine` command, which directs subsequent machine operations to the specified Agent.
- A set of machine operations to drive the application.
- A verification of the results of the machine operations.

You can use spawn to start a single thread for one machine, and then use successive spawn statements to start threads for other machines being tested. SilkTest scans for all spawn statements preceding a rendezvous statement and starts all the threads at the same time. However, the typical use of spawn is in a loop, like the following:

```
for each sMachine in lsMachine
  spawn        // start thread for each sMachine
```

```
    SetMachine (sMachine)
    DoSomething ()
  rendezvous
```

The preceding example achieves the same result when written as follows:

```
for each sMachine in lsMachine
  spawn
    [sMachine]DoSomething ()
  rendezvous
```

To use a spawn statement in tests that use TrueLog, use the `OPT_PAUSE_TRUELOG` option to disable TrueLog. Otherwise, issuing a spawn statement when TrueLog is enabled causes SilkTest to hang or crash.

Using Templates

Using the Parallel Template

This template is stored as `parallel.t` in the `Examples` subdirectory of the SilkTest installation directory. The code tests a single application that runs on an externally defined set of machines.

This multitestcase template accepts a list of machine names. The application whose main window is MyMainWin is invoked on each machine. The same operations are then performed on each machine in parallel. If any testcase fails, the multitestcase will be marked as having failed; however, a failed testcase within a thread does not abort the thread.

You can use this template by doing three edits:

- Include the file that contains your window declarations.
- Substitute the MainWin name of your application, which is defined in your MainWin window declaration, with the Mainwin name of the template, `MyMainWin`.
- Insert the calls to one or more tests, or to the main function, where indicated.

Use `myframe.inc`.

```
use "myframe.inc"
multitestcase MyParallelTest (LIST of STRING lsMachines)

  STRING sMachine

  // Connect to all machines in parallel:
  for each sMachine in lsMachines
    spawn
      SetUpMachine (sMachine, MyMainWin)
    rendezvous

  // Set app state of each machine, invoking if necessary:
  SetMultiAppStates()

  // Run testcases in parallel
  for each sMachine in lsMachines
    spawn
      SetMachine (sMachine)
      // Call testcase(s) or call main()
    rendezvous
```

ClientServer Template

This template is stored as `multi_cs.t` in the `Examples` subdirectory of the SilkTest installation directory. This testcase invokes the server application and any number of client applications, based on the list of machines passed to it, and runs the same function on all clients concurrently, after which the server will perform end-of-session processing.

You can use this template by doing the following edits:

- Include the files that contain your window declarations for both the client application and the server application.
- Substitute the MainWin name of your server application, which is defined in your MainWin window declaration, with the MainWin name of the template, `MyServerApp`.
- Substitute the MainWin name of your client application, which is defined in your MainWin window declaration, with the Mainwin name of the template, `MyClientApp`.
- Replace the call to `PerformClientActivity` with a function that you have written to perform client operations and tests.
- Replace the call to `DoServerAdministration` with a function that you have written to perform server administrative processing and/or cleanup.

```
use "myframe.inc"
multitestcase MyClientServerTest (STRING sServer, LIST of STRING lsClients)
  STRING sClient

  // Connect to server machine:
  SetUpMachine (sServer, MyServerApp)

  // Connect to all client machines in parallel:
  for each sClient in lsClients
    spawn
      SetUpMachine (sClient, MyClientApp)
  rendezvous

  // Set app state of each machine, invoking if necessary:
  SetMultiAppStates()

  // Run functions in parallel on each client:
  for each sClient in lsClients
    spawn
      // Make client do some work:
      [sClient] PerformClientActivity()
    rendezvous

  // Perform end-of-session processing on server application:
  [sServer] DoServerAdministration()
```

Testing Multiple Machines

Running Tests on One Remote Target

Use one of the following methods to specify that you want a script, suite, or test plan to run on a remote target instead of the host:

- Enter the name of the target Agent in the **Runtime Options** dialog box of the host. You also need to select a network protocol in the dialog box. If you have been testing a script by running SilkTest and the Agent on the same system, you can then test the script on a remote system without editing your script by using this method.
- Specify the target Agent's name by enclosing it within brackets before the script or suite name. For example `[Ohio]myscript.t`.
- You can select `(none)` in the **Runtime Options** dialog box of the host and then specify the name of the target Agent in a call to the `Connect` function in your script. For example, to connect to a machine named Ontario:

```
testcase MyTestcase ()
  Connect ("Ontario")
  // Call first testcase
  DoTest1 ()
  // Call second testcase
```

```
  DoTest2 ()
  Disconnect ("Ontario"
```

When you are driving only one remote target, there is no need to specify the current machine; all testcase code is automatically directed to the only connected machine.

When you use the multi-application support functions, you do not have to make explicit calls to `Connect`; the support functions issue these calls for you.

Running Tests Serially on Multiple Targets

To run your scripts or suites serially on multiple target machines, specify the name of the target Agent within the suite file. For example, the following code runs a suite of three scripts serially on two target machines named Ohio and Montana:

```
[Ohio] script1.t
[Ohio] script2.t
[Ohio] script3.t
[Montana] script1.t
[Montana] script2.t
[Montana] script3.t
```

Any spaces between the name of the target Agent and the script name are not significant.

Alternatively, to run testcases serially on multiple target machines, switch among the target machines from within the script, by using 4Test's `Connect` and `Disconnect` functions. For example, the following script contains a function named `DoSomeTesting` that is called once for each machine in a list of target machines, with the name of the target Agent as an argument:

```
testcase TestSerially ()
  STRING sMachine
  // Define list of agent names
  LIST OF STRING lsMachines = {...}
    "Ohio"
    "Montana"

  // Invoke test function for each name in list
  for each sMachine in lsMachines
    DoSomeTesting (sMachine)

  // Define the test function
  DoSomeTesting (STRING sMachine)
    Connect (sMachine)
    Print ("Target machine: {sMachine}")
    // do some testing...
    Disconnect (sMachine)
```

You will rarely need to run one test serially on multiple machines. Consider this example a step on the way to understanding parallel testing.

Specifying the Target Machine Driven by a Thread

While the typical purpose for a thread is to direct test operations to a particular test machine, you have total flexibility as to which machine is being driven by a particular thread at any point in time. For example, in the code below, the `spawn` statement starts a thread for each machine in a predefined list of test machines. The `SetMachine` command directs the code in that thread to the Agent on the specified machine. But the `["server"]` machine handle operator directs the code in the `doThis` function to the machine named *server*. The code following the `doThis` invocation continues to be sent to the *sMachine* specified in the `SetMachine` command.

```
for each smachine in lsMachine
  spawn // start thread for each sMachine
  SetMachine (sMachine)
  // ... code executed on sMachine
```

```
  ["server"]doThis() // code executed on "server"
  // ...continue with code for sMachine
rendezvous
```

While the machine handle operator takes only a machine handle, 4Test implicitly casts the string form of the Agent machine's name as a machine handle and so in the preceding example the machine name is effectively the same as a machine handle.

Specifying the Target Machine For a Single Command

To specify the target machine for a single command, use the machine handle operator on the command. For example, to execute the `SYS_SetDir` function on the target machine specified by the *sMachine1* variable, type `sMachine1->SYS_SetDir (sDir)`.

To allow you to conveniently perform system related functions (SYS_) on the host, you can preface the function call with the machine handle operator, specifying the globally defined constant *hHost* as the argument to the operator. For example, to set the working directory on the host machine to `c:\mydir`, type `hHost->SYS_SetDir ("c:\mydir")`.

You can use this syntax with a method call, for example `sMachine-> TextEditor.Search.Find.Pick`, but when invoking a method, this form of the machine handle must be the first token in the statement.

If you need to use this kind of statement, use the alternative form of the machine handle operator described below.

You can use the SetMachine function to change target machines for an entire block of code.

The hHost constant cannot be used in simple handle compares like hMyMachineHandle== hHost. This will never be TRUE. A better method is to use GetMachineName(hHost) and compare names. If hHost is used as an argument, it will refer to the "(local)" host not the target host.

Example

The following example shows valid and invalid syntax:

```
// Valid machine handle operator use
for each sMachine in lsMachine
  sMachine-> TextEditor.Search.Find.Pick

// Invalid machine handle operator use with method
if (sMachine->ProjX.DuplicateAlert.Exists())
  Print ("Duplicate warning on {sMachine} recipient.")
```

If you need to use this kind of statement, use the alternative form of the machine handle operator described below.

You can use the `SetMachine` function to change target machines for an entire block of code.

The *hHost* constant cannot be used in simple handle compares, like `hMyMachineHandle== hHost`. This will never be `TRUE`. A better method is to use `GetMachineName`(*hHost*) and compare names. If *hHost* is used as an argument, it will refer to the local host, not the target host.

Reporting Distributed Results

You can view test results in each of several formats, depending on the kind of information you need from the report. Each format sorts the results data differently, as follows:

Elapsed time	Sorts results for all threads and all machines in event order. This enables you to see the complete set of results for a time period and may give you a sense of the load on the server during that time period or may indicate a performance problem.

Machine	Sorts results for all threads running on one machine and presents the results in time-sorted order for that machine before reporting on the next machine.
Thread	Sorts results for all tests run under one thread and presents the results in time-sorted order for that thread before reporting on the next thread.

Alternative Machine Handle Operator

An alternative syntax for the machine handle operator is the bracket form, like the following example shows.

```
[hMachine] Any4TestFunctionCall ()
```

Example

To execute the `SYS_SetDir` function on the target machine specified by the string `sMachineA`, you do this:

```
[sMachineA] SYS_SetDir (sDir)
```

The correct form of the invalid syntax shown above is:

```
// Invalid machine handle operator use
if ([sMachine]ProjX.DuplicateAlert.Exists())
  Print ("Duplicate warning on {sMachine} recipient.")
```

To execute the `SYS_SetDir` function on the host machine, you can do the following:

```
[hHost] SYS_SetDir (sDir)
```

You can also use this form of the machine handle operator with a function that is not being used to return a value or with a method.

Example

```
for each sMachine in lsMachine
     [sMachine] FormatTest7 ()
```

Example

```
for each sMachine in lsMachine
  [sMachine] TextEditor.Search.Find.Pick
```

Testing Clients Concurrently

In concurrent testing, SilkTest executes one function on two or more clients at the same time. This topic demonstrates one way to perform the same tests concurrently on multiple clients.

For example, suppose you want to initiate two concurrent database transactions on the same record, and then test how well the server performs. To accomplish this, you need to change the script presented in *Testing Clients Plus Server Serially* to look like this:

```
testcase TestConcurrently ()
  Connect ("server")
  Connect ("client1")
  Connect ("client2")
  DoSomeSetup ("server")    // initialize server first
  Disconnect ("server")     // testcase is thru with server

  spawn                     // start thread for client1
    UpdateDatabase ("client1")
  spawn                     // start thread for client2
    UpdateDatabase ("client2")
```

```
  rendezvous                // synchronize
  Disconnect ("client1")
  Disconnect ("client2")

  DoSomeSetup (STRING sMachine)   // define server setup
    HTIMER hTimer
    hTimer = TimerCreate ()
    TimerStart (hTimer)
    SetMachine (sMachine)
    // code to do server setup goes here
    TimerStop (hTimer)
    Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
    TimerDestroy (hTimer)

  UpdateDatabase (STRING sMachine)   // define update test
    HTIMER hTimer
    hTimer = TimerCreate ()
    TimerStart (hTimer)
    SetMachine (sMachine)
    // code to update database goes here
    TimerStop (hTimer)
    Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
    TimerDestroy (hTimer)
```

An alternative but equivalent approach is to use the parallel statement in place of the spawn and rendezvous:

```
testcase TestConcurrently2 ()
  Connect ("server")
  Connect ("client1")
  Connect ("client2")

  DoSomeSetup ("server")
  Disconnect ("server")

  parallel
    UpdateDatabase ("client1") // thread for client1
    UpdateDatabase ("client2") // thread for client2
                               // automatic synchronization
    Disconnect ("client1")
    Disconnect ("client2")

  DoSomeSetup (STRING sMachine)
    HTIMER hTimer
    hTimer = TimerCreate ()
    TimerStart (hTimer)
    SetMachine (sMachine)
    // code to do server setup goes here
    TimerStop (hTimer)
    Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
    TimerDestroy (hTimer)

  UpdateDatabase (STRING sMachine)
    HTIMER hTimer
    hTimer = TimerCreate ()
    TimerStart (hTimer)
    SetMachine (sMachine)
    // code to update database goes here
    TimerStop (hTimer)
    Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
    TimerDestroy (hTimer)
```

If you use variables to specify different database records for each client's database transactions, you can use the above techniques to guarantee parallel execution without concurrent database accesses.

Testing Clients Plus Server Serially

In a client/server application, the server and its clients typically run on different target machines. This topic explains how to build tests that test the server and its clients in a serial fashion. In this scenario, the `SetMachine` function switches among the target machines on which the server and its clients are running. The following script fragment tests a client/server database application in the following steps:

1. Connect to three target machines, which are server, client1, and client2.
2. Call the `DoSomeSetup` function, which calls `SetMachine` to make "server" the current target machine, and then perform some setup.
3. Call the `UpdateDatabase` function once for each client machine. The function sets the target machine to the specified client, then does a database update. It creates a timer to time the operation on this client.
4. Disconnect from all target machines.

Example

This example shows how you direct sets of testcase statements to particular machines. If you were doing functional testing of one application, you might want to drive the server first and then the application. However, this example is not realistic because it does not show the support necessary to bring the different machines to their different application states and to recover from a failure on any machine.

```
testcase TestClient_Server ()
  Connect ("server")
  Connect ("client1")
  Connect ("client2")
  DoSomeSetup ("server")
  UpdateDatabase ("client1")
  UpdateDatabase ("client2")
  DisconnectAll ()

DoSomeSetup (STRING sMachine)
  HTIMER hTimer
  hTimer = TimerCreate ()
  TimerStart (hTimer)
  SetMachine (sMachine)
  // code to do server setup goes here
  TimerStop (hTimer)
  Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
  TimerDestroy (hTimer)

UpdateDatabase (STRING sMachine)
  HTIMER hTimer
  hTimer = TimerCreate ()
  TimerStart (hTimer)
  SetMachine (sMachine)
  // code to update database goes here
  TimerStop (hTimer)
  Print ("Time on {sMachine} is: {TimerStr (hTimer)}")
  TimerDestroy (hTimer)
```

Testing Databases

You may be testing a distributed application that accesses a database or you may be directly testing database software. In either of these cases, you might want to manipulate the database directly from SilkTest for several purposes:

- To exercise certain database functions that are present in a GUI that runs directly on the server machine and is not a client application. For example, administrative functions used for setting up the database.
- To set the server database to a known state.
- To verify an application's database results without using the application.
- To read information from the database to use as input to a testcase.

SilkTest can drive a server application's GUI by means of the SilkTest Agent exactly as it drives a client application. In addition, the database tester provides direct access, using SQL, from a test script to any database supported by ODBC drivers. These database functions enable you to read and write database records without using the client application. Thus, you can verify client test results without assuming the ability of the client to do that verification.

In addition to using the SQL functions in your tests, you can also use these functions to help manage your testing process as follows:

- Maintain a bug database, updating it with the results of your testing.
- Manage your test data in a database instead of in a text file.

The database functions, among other things, allow you to connect to a database, submit an SQL statement, read data from the selected record(s) if the SQL statement was SELECT, and subsequently disconnect from the database. About a dozen of these functions allow you to access your database's catalog tables.

The functions that support these operations begin with the letters "DB_".

Multi-Machine Testing in a Terminal Server Environment

This functionality is available only for projects or scripts that use the Classic Agent.

Note: This functionality is tested only for C++ applications. Whether any other extension will work is unconfirmed. If a non-C++ environment does not work, then it is still considered a non-supported environment and is not something that will be addressed by technical support.

Terminal Server

Terminal Server is an optional setup of Windows where the server is used in a similar fashion as a Unix server. In a network of this type you have a server with a lot of memory to serve many workstations. Each workstation has its own operating system and is connected through TCP/IP to the server machine. Each client is required to have only one program installed; the terminal client.

The terminal client is a program that displays a complete Windows desktop, including a taskbar, just like the one you see when you run Windows. Using the mouse and keyboard you are able to use this Desktop to start and use Windows Applications like Word, PowerPoint and SilkTest. When these applications are running, they are not using the CPU or memory of the client machines, but are running on the server machines. The display of the desktop however is being set to the terminal client programs. They do this by sending many compressed images in cartoon fashion through TCP/IP from the server to the terminal emulators, which in turn display the images and make it appear as though the Desktop and the applications are running on the client machine.

Each terminal emulator has its own virtual mouse and keyboard port. You can have several different Terminal Clients running on the same machine and each window will have its own mouse pointer. When you use the physical mouse on the terminal client, the virtual mouse reacts to the commands.

Installing SilkTest on the Controller Machine

In a Terminal Server environment the copy of SilkTest is only installed on the server machine. Only one copy is needed. When installing the product, you must follow the Terminal Server instructions for installing the application in 'multi user' mode. This is done from the Add/Remove Programs feature found in the Control Panel. When this is complete, you can start SilkTest from a terminal client and create and run tests as you do when SilkTest is installed on one machine.

Setting Up the Terminal Server Clients

In order to have multiple terminal clients running SilkTest or other Agents, some configuration must take place. If you want to be able to run test scenarios in different terminal client windows from a single point of control, each client needs to start its own Agent process. Each Agent must be configured for network use. You can either use the NetBios protocol and give each Agent instance a separate name, or use the TCP/IP protocol and assign each Agent instance a different port.

SilkTest does not support multiple-user sessions for a single target system. Using standalone SilkTest you can only service one Agent on a particular machine. However, using SilkPerformer as controller for SilkTest scripts, you can also service multiple user/Agent sessions on a single system (SilkPerformer Gui level testing).

Starting Agents

SilkTest has a -p option for `Agent.exe` that lets you specify a port number for the Agent. Therefore, it is recommended that you use TCP/IP protocol and start Agents using `agent -p <unique port number>`.

Testing Multiple Applications

Overview of Multi-Application Testing

SilkTest can easily drive multiple different applications simultaneously. Thus you can bring a server's database to a known state at the same time you are bringing multiple instances of the client application to their base state window. Likewise, you can drive a server database with several different client applications at the same time.

The essential difference between single-application and multi-application testing is clearly the difference between "one" and "many." When the following entities in a test case are greater than one, they need special consideration and support functions found in SilkTest:

- Agent names.
- Application main window names.
- Sets of application states associated with each main window name.

Multi-machine testing requires that you map both the name of an application and all application states for that application to the machine on which it will be tested. This makes it possible for you to direct test operations to the right machines, and it enables SilkTest to automatically set the machines to the proper application state before a test is run, and to clean up after a test has failed.

Test Case Structure in a Multi-Application Environment

This topic describes SilkTest components that enable concurrent testing of more than one application. For example, there are functions that make it possible to drive both the client application and the client's server from SilkTest, to set each to its base state, and to recover each if it fails. Compare with the test case structure of a single-application environment.

The multi-application environment uses the same `defaults.inc` file as does the single-application environment. However, when you define a function as a multitestcase, 4Test uses functions defined in the `cs.inc` file to invoke functions in `defaults.inc`. Thus, it can pass the appropriate application states or base states to these functions, depending on the requirements of a particular test machine.

Instead of preceding the test case function declaration with the keyword testcase, you must use the keyword multitestcase to give your test case the multi-application recovery system. For additional information, see *Invoking a Test Case: Multi-Application Environment*.

`cs.inc` is an automatically included file that contains functions used only in the multi-application environment. For additional information about this file and the functions that it contains, see `cs.inc`. You may need to include other files also; For additional information, see *Other Window Declarations*.

Invoking a Test Case Multi-Application Environment

The keyword for a test case declaration is different when you are performing distributed testing. In the single-application environment, you invoke a test case with no arguments or you may specify an application state function. However, in a multi-application environment, instead of preceding the test case function declaration with the keyword testcase, you must use the keyword multitestcase to give your test case the multi-application recovery system.

Declaring a function as a multitestcase gives that function the ability to invoke functions declared with the keyword testcase. A multitestcase thus can be viewed as a wrapper for stand-alone test cases; it provides a means of assigning tests to particular machines and lets you invoke previously written test cases from the multi-test case file by simply adding a use statement to the file to include the test case definitions.

When you are using multi-application environment support, you can pass the test case the names of the machines to be tested during that execution of the test case, but not the application state function. In a multi-application environment, one test case can use multiple application states; you specify these in the required code at the beginning of the test case.

Test Case Structure in a Single-Application Environment

The code that implements a test case for a single application is similar to that of a test case for applications on multiple separate machines in a client/server environment.

This topic summarizes the structure of the single-application version and some SilkTest components used to implement it. You can compare the structure with the support code needed for running multiple applications.

The include file `defaults.inc` implements the recovery system for a single application test. For information about the `DefaulBaseState` function and the functions that are contained within `defaults.inc`, see *defaults.inc*.

Your test case needs certain definitions that other test cases in your testing program will also need. These include:

- Window declarations
- Application states
- Utility functions

Placing these general purpose definitions in an include file, or several smaller files, saves repetitive coding. When you use SilkTest to record window declarations and application states, SilkTest names the generated file `frame.inc`.

Window Declarations for Multi-Application Testing

In the client/server environment, unlike the stand-alone environment, you can test two or more different applications at the same time. For example, you could run the functional tests for application "A" on a Microsoft Windows 2000 machine and a Microsoft Windows XP machine at the same time that you are running the functional tests for application "B" on both Windows 2000 and Windows XP machines. The include files that you must generate may therefore have to take into consideration different platforms and/or different applications.

When you are driving two or more applications from SilkTest, you need separate window declarations for each different application. You must be certain that your main window declaration for each separate application is unique. If the same application is running on different platforms concurrently, you may need to use GUI specifiers to specialize the window declarations. 4Test will identify a window declaration statement, that is preceded by a GUI specifier, as being true only on the specified GUI.

In addition, you may find that the operations needed to establish a particular application state are slightly different between platforms. In this case, you just record application states for each platform and give them names that identify the state and the GUI for your convenience.

Recording window declarations on a client machine that is not the host machine, requires that you operate both SilkTest on the host machine and the application on its machine at the same time. You record window declarations and application states in much the same way for a remote machine as for an application running in the SilkTest host machine. The primary difference is that you start the recording operation by selecting Test Frame in SilkTest on the host system and you do the actual recording of application operations on the remote system.

Note: Remote recording is available only for projects or scripts that use the Classic Agent.

If you have two or more applications being tested in parallel, you need to have two or more sets of window declarations. You must have window declarations, and application states, if needed, for each different application. When recording window declarations and application states on a remote machine, you will find it convenient to have the machine physically near to your host system.

Concurrency Test Example Code

The concurrency test example is designed to allow any number of test machines to attempt to access a server database at the same time. This tests for problems with concurrency, such as deadlock or out-of-sequence writes.

This example uses only one application. However, it is coded in the style required by the multi-application environment because you will probably want to use an Agent to start and initialize the server during this type of test. There is no requirement in the client/server environment that you use the single-application style of testcase just because you are driving only one application. For consistency of coding style, you will probably find it convenient to always use the multi-application files and functions.

For detailed information on the code example, see *Concurrency Test Explained*.

```
const ACCEPT_TIMEOUT = 15
multitestcase MyTest (LIST OF STRING lsMachine)
  STRING sMachine
  INTEGER iSucceed
  STRING sError

  for each sMachine in lsMachine
    SetUpMachine (sMachine, Personnel)
    SetMultiAppStates ()

  /*** HAVE EACH MACHINE EDIT THE SAME EMPLOYEE ***/
  for each sMachine in lsMachine
    spawn

      /*** SET THE CURRENT MACHINE FOR THIS THREAD ***/
      SetMachine (sMachine)

      /*** EDIT THE EMPLOYEE RECORD "John Doe" ***/
      Personnel.EmployeeList.Select ("John Doe")
      Personnel.Employee.Edit.Pick ()

      /*** CHANGE THE SALARY TO A RANDOM NUMBER BETWEEN
      50000 AND 70000 ***/
      Employee.Salary.SetText ([STRING] RandInt (50000, 70000))
  rendezvous

  /*** ATTEMPT TO HAVE EACH MACHINE SAVE THE EMPLOYEE RECORD ***/
  for each sMachine in lsMachine
    spawn

      /*** SET THE CURRENT MACHINE FOR THIS THREAD ***/
      SetMachine (sMachine)

      /*** SELECT THE OK BUTTON ***/
```

```
      Employee.OK.Click ()

      /*** CHECK IF THERE IS A MESSAGE BOX ***/
      if (MessageBox.Exists (ACCEPT_TIMEOUT))
        SetMachineData (NULL, "sMessage",
          MessageBox.Message.GetText ())
        MessageBox.OK.Click ()
        Employee.Cancel.Click ()
      else if (Employee.Exists ())
        AppError ("Employee dialog not
          dismissed after {ACCEPT_TIMEOUT} seconds")
  rendezvous

  /*** VERIFY THE OF NUMBER OF MACHINES WHICH SUCCEEDED ***/
  iSucceed = 0
  for each sMachine in lsMachine
    sError = GetMachineData (sMachine, "sMessage")
    if (sMessage == NULL)
      iSucceed += 1
    else
      Print ("Machine {sMachine} got message '{sMessage}'")

  Verify (iSucceed, 1, "number of machines that succeeded")
```

Concurrency Test Explained

Before you record and/or code your concurrency test, you record window declarations that describe the elements of the application's GUI. These are placed in a file named `frame.inc`, which is automatically included with your test case when you compile. Use SilkTest to generate this file because SilkTest does most of the work.

The following code sample gives just those window declarations that are used in the *Concurrency Test Example*:

```
window MainWin Personnel
  tag "Personnel"
  PopupList EmployeeList
  Menu Employee
    tag "Employee"
  MenuItem Edit
    tag "Edit"
  // ...

window DialogBox Employee
  tag "Employee"
  parent Personnel
  TextField Salary
    tag "Salary"
  PushButton OK
    tag "OK"
  // ...
```

The following explanation of the *Concurrency Test Example* gives the testing paradigm for a simple concurrency test and provides explanations of many of the code constructs. This should enable you to read the example without referring to the Help. There you will find more detailed explanations of these language constructs, plus explanations of the constructs not explained here. The explanation of each piece of code follows that code.

```
const ACCEPT_TIMEOUT = 15
```

The first line of the testcase file defines the timeout value (in seconds) to be used while waiting for a window to display.

```
multitestcase MyTest (LIST OF STRING lsMachine)
```

The test case function declaration starts with the multitestcase keyword. It specifies a LIST OF STRING argument that contains the machine names for the set of client machines to be tested. You can implement and maintain this list in your test plan, by using the test plan editor. The machine names you use in this list are the names of the Agents of your target machines.

```
for each sMachine in lsMachine
  SetUpMachine (sMachine, Personnel)
```

To prepare your client machines for testing, you must connect SilkTest to each Agent and, by means of the Agent, bring up the application on each machine. In this example, all Agents are running the same software and so all have the same MainWin declaration and therefore just one test frame file. This means you can initialize all your machines the same way; for each machine being tested, you pass to SetUpMachine the main window name you specified in your test frame file. The SetUpMachine function issues a Connect call for each machine. It associates the main window name you specified (Personnel) with each machine so that the DefaultBaseState function can subsequently retrieve it.

```
SetMultiAppStates ()
```

The SetMultiAppStates function reads the information associated with each machine to determine whether the machine needs to be set to an application state. In this case no application state was specified (it would have been a third argument for SetUpMachine). Therefore, SetMultiAppStates calls the DefaultBaseState function for each machine. In this example, DefaultBaseState drives the Agent for each machine to open the main window of the Personnel application. This application is then active on the client machine and 4Test can send testcase statements to the Agent to drive application operations.

```
 for each sMachine in lsMachine
  spawn
    // The code to be executed in parallel by
    // all machines... (described below)
rendezvous
```

Because this is a concurrency test, you want all client applications to execute the test at exactly the same time. The spawn statement starts an execution thread in which each statement in the indented code block runs in parallel with all currently running threads. In this example, a thread is started for each machine in the list of machines being tested. 4Test sends the statements in the indented code block to the Agents on each machine and then waits at the rendezvous statement until all Agents report that all the code statements have been executed.

The following is the code defined for the spawn statement:

```
// The code to be executed in parallel by
// all machines:
SetMachine (sMachine)
Personnel.EmployeeList.Select ("John Doe")
Personnel.Employee.Edit.Pick ()
Employee.Salary.SetText
[STRING] RandInt (50000, 70000))
```

Each thread executes operations that prepare for an attempt to perform concurrent writes to the same database record. The SetMachine function establishes the Agent that is to execute the code in this thread. The next two statements drive the application's user interface to select John Doe's record from the employee list box and then to pick the Edit option from the Employee menu. This opens the Employee dialog box and displays John Doe's employee record. The last thread operation sets the salary field in this dialog box to a random number. At this point the client is prepared to attempt a write to John Doe's employee record. When this point has been reached by all clients, the rendezvous statement is satisfied, and 4Test can continue with the next script statement.

```
for each sMachine in lsMachine
  spawn
    SetMachine (sMachine)
    Employee.OK.Click ()
    if (MessageBox.Exists (ACCEPT_TIMEOUT))
      SetMachineData (NULL, "sMessage",
      MessageBox.Message.GetText ())
```

```
      MessageBox.OK.Click ()
      Employee.Cancel.Click ()
    else if (Employee.Exists ())
      AppError ("Employee dialog not dismissed
        after {ACCEPT_TIMEOUT} seconds")
rendezvous
```

Now that all the clients are ready to write to the database, the script creates a thread for each client, in which each attempts to save the same employee record at the same time. There is only one operation for each Agent to execute: `Employee.OK.Click`, which clicks the **OK** button to commit the edit performed in the previous thread.

The test expects the application to report the concurrency conflict with message boxes for all but one client and for that client to close its dialog box within 15 seconds. The if-else construct saves the text of the message in the error message box by means of the SetMachineData function. It then closes the message box and the Employee window so that the recovery system will not report that it had to close windows. This is good practice because it means fewer messages to interpret in the results file.

The "else if" section of the if-else checks to see whether the Employee window remains open, presumably because it is held by a deadlock condition; this is a test case failure. In this case, the AppError function places the string "***ERROR:" in front of the descriptive error message and raises an exception; all Agents terminate their threads and the test case exits.

```
iSucceed = 0
for each sMachine in lsMachine
  sMessage = GetMachineData (sMachine, "sMessage")
  if (sMessage == NULL)
    iSucceed += 1
  else
    Print ("Machine {sMachine} got message '{sMessage}'")
Verify (iSucceed, 1, "number of machines that succeeded")
```

The last section of code evaluates the results of the concurrency test in the event that all threads completed. If more than one client successfully wrote to the database, the test actually failed.

GetMachineData retrieves the message box message (if any) associated with each machine. If there was no message, iSucceed is incremented; it holds the count of "successes." The Print function writes the text of the message box to the results file for each machine that had a message box. You can read the results file to verify that the correct message was reported. Alternatively, you could modify the test to automatically verify the message text.

The Verify function verifies that one and only one machine succeeded. If the comparison in the Verify function fails, Verify raises an exception. All exceptions are listed in the results file.

Notification Test Example Code (1 of 2)

This functionality is available only for projects or scripts that use the Classic Agent.

This topic contains the complete test case file for a single-user notification test. It shows a testing technique for a type of communication frequently used in client/server applications. *Notification Test Example Code (2 of 2)* shows a notification test between two users running their own copies of the client application. This illustrates doing the simplest case first and then adding the next level of complexity when you go from one user to two users. For additional information on the testing technique, see *Notification Test Example Explained (1 of 2)*.

```
// ccmail.t
use "ccmail.inc"
LogMeIn()
  LogInUser(GetMachineData( NULL, "Username" ),
    GetMachineData( NULL, "Password" ) )

//-------------------------------------------------------------
multitestcase SingleUserNotification ( STRING sMachine1 optional )
```

```
  if( sMachine1 == NULL )
    sMachine1 = "(local)"

  //=== MULTI-APPLICATION SETUP SECTION ==================//
  SetUpMachine( sMachine1, CcMail, "EnsureInBoxIsEmpty" )
  SetMachineData( sMachine1, "Username", "QAtest1" )
  SetMachineData( sMachine1, "Password", "QAtest1" )
  SetMultiAppStates()

  //=== TEST BEGINS HERE ================================//
  SetMachine( sMachine1 )
  SimpleMessage( "QAtest1", "Message to myself", "A message to myself" )
  Verify( CcMailNewMailAlert.Exists( NOTIFICATION_TIMEOUT ), TRUE )
  Verify( CcMailNewMailAlert.IsActive(), TRUE, "ALERT" )
  CcMailNewMailAlert.OK.Click()
  CcMail.xWindow.GoToInbox.Pick ()
  Verify( CcMail.Message.DeleteMessage.IsEnabled(), TRUE,
    "MESSAGE WAITING" )
```

Utility function

```
void SimpleMessage (STRING sRecipient, STRING sSubject,
  STRING sBody)
CcMail.Message.NewMessage.Pick()

NewMessage.MailingLabel.Recipient.SetText (sRecipient)
NewMessage.MailingLabel.Recipient.TypeKeys ("<Enter>")
NewMessage.MailingLabel.Recipient.TypeKeys ("<Enter>")
NewMessage.MailingLabel.SubjectField.SetText (sSubject)
NewMessage.MailingLabel.SubjectField.TypeKeys ("<Enter>")
NewMessage.EditBody.Body.TypeKeys (sBody)
NewMessage.EditBody.Body.TypeKeys ("<Ctrl-s>")
```

This function uses standard methods on Ccmail window components, defined in `ccmail.inc`, to do the following:

1. Pick the NewMessage item from the Message menu.
2. Enter the string in argument one into the Recipient field and press the **Enter** key twice to move to the Subject field.
3. Enter the string in argument two into the Subject field and press **Enter** to move to the message body portion of the window (EditBody.Body).
4. Type the string in argument three into the Body field and type **Ctrl + s** to send the message.

The following block of code verifies the results of the test.

```
Verify(CcMailNewMailAlert.Exists(NOTIFICATION_TIMEOUT),
  TRUE )
Verify(CcMailNewMailAlert.IsActive(), TRUE, "ALERT")
CcMailNewMailAlert.OK.Click()
CcMail.xWindow.GoToInbox.Pick ()
Verify(CcMail.Message.DeleteMessage.IsEnabled(), TRUE,
  "MESSAGE WAITING")
```

The above code does the following:

1. Verifies that the sent message was received, as indicated by the **NewMailAlert** message box. The `NOTIFICATION_TIMEOUT` value causes the `Verify` function to wait for that period of time for the window to exist. If the timeout value is reached, the `Verify` raises an exception.
2. Verify that the dialog box **CcMailNewMailAlert** is active.
3. If the `Verify` executes without an exception, click on the **OK** button in the **CcMailNewMailAlert** dialog box.
4. Pick the **GoToInbox** menu item from the Window menu.

5. Verify that a message exists in the Inbox by checking to see that the Message menu has its **DeleteMessage** menu item enabled. If the menu item is not enabled, there is no message in the Inbox and the `Verify` function raises an exception.

- This script continues in *Notification Test Example Code (2 of 2)*.

Notification Test Example Explained (1 of 2)

This functionality is available only for projects or scripts that use the Classic Agent.

The first line in the test case file is a comment that lists the name of the file holding this code.

```
// csmail.t
```

The next line is an include statement. The explanations for each fragment of code follow that code.

```
use "ccmail.inc"
```

The `ccmail.inc` file is defined for this test case. It contains the window declarations for the application in addition to application state definitions and definitions for general-purpose utility functions also needed by other test cases designed for this application. You can find the `ccmail.inc` file in the SilkTest Examples directory. Code fragments from that file are included as needed in this discussion.

```
LogMeIn()
  LogInUser(GetMachineData(NULL, "Username"),
    GetMachineData(NULL, "Password") )
```

The utility function LogMeIn is called by the invoke method for the CC Mail main window, called `CcMail`. The LogInUser function is defined in `ccmail.inc`. The machine data that LogInUser retrieves for its arguments gets established by each test before the application state function for each machine is invoked.

```
multitestcase SingleUserNotification (STRING sMachine1)
```

The function declaration for the test case passes in the name of the SilkTest Agent for the machine on which the application is running.

```
if(sMachine1 == NULL)
  sMachine1 = "(local)"
```

This if statement if statement allows you to invoke the test case without specifying a machine name when you want to run on the local machine.

```
SetUpMachine(sMachine1, CcMail, "EnsureInBoxIsEmpty")
```

The SetUpMachine function provides the name of the main application window, `CcMail`, and the application state (`EnsureInBoxIsEmpty`) to be established by SilkTest. `EnsureInBoxIsEmpty` is defined in `ccmail.inc`. This statement is part of the standard setup code for multi-application tests. The standard multi-application setup code is documented in *template.t Explained* and *Concurrency Test Example Code*. The setup code in this test case is essentially the same.

This is a single-user test case and therefore does not actually need the setup methodology required by a multi-application test. However, since client/server testing is frequently multi-application testing, all the example test cases use the multi-application coding methods. We recommend that you also follow this practice, since consistency of testing styles reduces coding errors in your test cases.

One difference for this test case is that this is an application that requires the user to log in. Therefore the following code fragment provides the user name and password for the application under test:

```
SetMachineData (sMachine1, "Username", "QAtest1")
SetMachineData (sMachine1, "Password", "QAtest1")
```

These statements associate two pieces of information, named "Username" and "Password," with the specified machine. In both cases the value of the associated information is the same, "QAtest1." Now that this information is available to the application state function, that function can log the user in. This will happen as a result of the next statement.

```
SetMultiAppStates()
```

In this test, `SetMultiAppStates` function will actually only set the application state for the one machine.

```
SimpleMessage ("QAtest1", "Message to myself",
  "A message to myself")
```

The above line invokes the utility function from `ccmail.inc`, which sends the short message to the local machine.

Notification Test Example Code (2 of 2)

This functionality is available only for projects or scripts that use the Classic Agent.

This is the complete test case file for a two-user notification test. It shows the next level of complexity in testing client/server notification operations. For additional information on the testing technique, see *Notification Example 2 Explained*.

```
//-------------------------------------------------------------
// This testcase logs in as user QAtest1 on the first machine,
// and logs in as user QAtest2 on the second machine; then
// sends a message from the user on the first machine to the
// user on the second machine; it then switches to the second
// machine and waits to be notified that new mail has arrived.
//

multitestcase TwoUserNotification ( STRING sMachine1, STRING sMachine2 )

  //=== MULTI-APPLICATION SETUP SECTION ==================//
  SetUpMachine( sMachine1, CcMail )
  SetUpMachine( sMachine2, CcMail, "EnsureInBoxIsEmpty" )
  SetMachineData( sMachine1, "Username", "QAtest1" )
  SetMachineData( sMachine1, "Password", "QAtest1" )
  SetMachineData( sMachine2, "Username", "QAtest2" )
  SetMachineData( sMachine2, "Password", "QAtest2" )
  SetMultiAppStates()

  //=== TEST BEGINS HERE ================================//
  //-------------------------------------------------------
  // Switch to the first machine:
  SetMachine( sMachine1 )

  // Send mail from user 1 to user 2
  SimpleMessage("QAtest2", "Message to user 2", "Message from me to you.")

  //-------------------------------------------------------
  // Switch to the second machine:
  SetMachine( sMachine2 )

  // Wait for notification to occur, then acknowledge it:
  Verify( CcMailNewMailAlert.Exists( NOTIFICATION_TIMEOUT ), TRUE )
  Verify( CcMailNewMailAlert.IsActive(), TRUE, "ALERT" )
  CcMailNewMailAlert.OK.Click()

  // Refresh the In box and check that a message is waiting there:
  CcMail.xWindow.GoToInbox.Pick ()
  Verify( CcMail.Message.DeleteMessage.IsEnabled(), TRUE,
    "MESSAGE WAITING" )
```

Notification Test Example Explained (2 of 2)

This functionality is available only for projects or scripts that use the Classic Agent.

The code in this two-user notification test is much the same as the code in the single-user example, except that the test is distributed across two CcMail applications. Thus the primary differences in this example are in the program flow.

Note: The described actions are carried out sequentially rather than concurrently.

The following actions are carried out by the code in the two-user notification test:

Before the test starts

1. The SetUpMachine function is carried out on two machines; the first machine defaults to the base state, but the second machine specifies an application state that ensures that its InBox is empty.
2. The Username and Password values for both machines are set.
3. The SetMultiAppStates function is invoked for both machines.

Note: This function will set different application states for the two machines.

On Machine 1:

1. The SetMachine function specifies that Machine 1 should receive the next operation.
2. A simple message is sent to Machine 2.

On Machine 2:

1. Verify that the **Alert** dialog box exists.
2. Verify that the **Alert** dialog box is active. If it is not, the exception's error message will be `Verify ALERT failed...`.
3. If the **Alert** dialog box has opened, dismiss it by clicking **OK**.
4. Refresh the InBox by picking the Inbox choice from CcMail's Window menu.
5. Verify that the Message menu's DeleteMessage menu item is enabled, proving that the message is in the Inbox. If Verify fails, which means the menu item is not enabled, the exception's error message will read, `Verify MESSAGE WAITING failed...`.

Code for template.t

This fragment of an example testcase shows the required code with which you start a multi-application test case. It connects SilkTest to all the machines being tested and brings each to its first screen. This is just a template; you must tailor your code to fit your actual needs. For information on the significance of each line of code, see *Template.t Explained*.

```
multitestcase MyTest (STRING sMach1, STRING sMach2)
  SetUpMachine (sMach1, MyFirstApp, "MyFirstAppState")
  SetUpMachine (sMach2, MySecondApp, "MySecondAppState")
  SetMultiAppStates ()
    spawn
      SetMachine (sMach1)
      // Here is placed code that drives test operations

    spawn
      SetMachine (sMach2)
      // Here is placed code that drives test operations

  rendezvous
  // "..."
```

template.t Explained

The following line of code in *Code for template.t* is the first required line in a multi-application test case file. It is the test case declaration.

Note: The code does not pass an application state as in the stand-alone environment.

```
multitestcase MyTest (STRING sMach1, STRING sMach2)
```

In the multi-application environment the arguments to your test case are names of the machines to be tested; you specify application states inside the test case. You can code the machine names arguments as you like. For example, you can pass a file name as the only argument, and then, in the test case, read the names of the machines from that file. Or you can define a LIST OF HMACHINE data structure in your test plan, if you are using the test plan editor, to specify the required machines and pass the name of the list, when you invoke the test case from the test plan. This template assumes that you are using a test plan and that it passes the Agent names when it invokes the test case. For this example, the test plan might specify the following:

```
Mytest ("Client1", "Client2")
```

The next two code lines are the first required lines in the test case:

```
SetUpMachine (sMach1, MyFirstApp, "MyFirstAppState")
SetUpMachine (sMach2, My2ndApp, "My2ndAppState")
```

You must execute the `SetUpMachine` function for every client machine that will be tested. For each `SetUpMachine` call, you specify the application to be tested, by passing the name of the main window, and the state to which you want the application to be set, by passing the name of the application state if you have defined one.

The `SetUpMachine` function issues a `Connect` call for a machine you want to test and then configures either the base state or a specified application state.

It does this as follows:

- It associates the client application's main window name with the specified machine so that the `DefaultBaseState` function can subsequently retrieve it to set the base state.
- It associates the name of the application's base state, if one is specified, with the specified machine so that the `SetMultiAppStates` function can subsequently retrieve it and set the application to that state at the start of the test case.

The first argument for `SetUpMachine` is the machine name of one of your client machines. The second argument is the name you supply in your main window declaration in your test frame file, `frame.inc`. For this example, the `frame.inc` file specifies the following:

```
window MainWin MyFirstApp
```

Because this template specifies two different applications, it requires two different test frame files.

The third argument is the name you provide for your application state function in your appstate declaration for this test. For this example, the appstate declaration is the following:

```
appstate MyFirstAppState () based on MyFirstBaseState
```

The appstate declaration could also be of the form:

```
appstate MyFirstBaseState ()
```

Although the `DefaultBaseState` function is designed to handle most types of GUI-based applications, you may find that you need to define your own base state. It would be the application state that all your tests for this application use. In this case, you would still pass this application state to `SetUpMachine` so that your application would always be brought to this state at the start of each test case.

This template specifies two application states for generality. You would not use an application state if you wanted to start from the main window each time. If you have a number of tests that require you to bring the application to the same state, it saves test-case code to record the application state once, and pass its name to `SetUpMachine`. You will probably place your application state declarations in your test frame file.

```
SetMultiAppStates ()
```

The `SetMultiAppStates` function must always be called, even if the test case specifies no application state, because `SetMultiAppStates` calls the `DefaultBaseState` function in the absence of an

appstate declaration. `SetMultiAppStates` uses the information that `SetUpMachine` associated with each connected machine to set potentially different application states or base states for each machine.

```
 spawn
  SetMachine (sMach1)
  // Here is placed code that drives test operations
```

The spawn statement starts an execution thread, in which each statement in the indented code block below it runs in parallel with all currently running threads. There is no requirement that your test case should drive all your test machines at the same time, however, this is usually the case. The `SetMachine` function directs 4Test to execute this thread's code by means of the Agent on the specified machine. This thread can then go on to drive a portion, or all, of the test operations for this machine.

```
 spawn
  SetMachine (sMach2)
  // Here is placed code that drives test operations
rendezvous
// "..."
```

The second spawn statement starts the thread for the second machine in this template. The rendezvous statement blocks the execution of the calling thread until all threads spawned have completed. You can use the rendezvous statement to synchronize machines as necessary before continuing with the test case.

defaults.inc

The `defaults.inc` file is provided by SilkTest and implements the recovery system for a single application test. That is, it contains the `DefaultBaseState` function that performs any cleanup needed after an operation under test fails and returns the application to its base state.

You can define a base state function to replace the `DefaultBaseState` function by defining an application state without using the basedon keyword. This creates an application state that 4Test executes instead of the `DefaultBaseState` function.

The `defaults.inc` file contains six other functions that 4Test automatically executes unless you define functions that replace them:

DefaultScriptEnter	A null function, allows you to define a `ScriptEnter` function, as discussed below.
DefaultScriptExit (BOOLEAN bException)	Logs an exception to the results file when a script exits because of an exception.
DefaultTestcaseEnter	Executes the `SetAppState` function. If you have specified an application state for this test case, the `SetAppState` function brings your test application to that state. If you have no application state defined, `SetAppState` brings the application to the base state (if necessary).
DefaultTestcaseExit (BOOLEAN bException)	Logs an exception to the results file when a test case exits because of an exception. The function then executes the `SetBaseState` function, which calls a base state function that you have defined or the `DefaultBaseState` function.
DefaultTestPlanEnter	A null function, allows you to define `TestPlanEnter`, as discussed below, to allow logging of results.
DefaultTestPlanExit (BOOLEAN bException)	A null function, allows you to define `TestPlanExit`, as discussed below, to allow logging of results.

The word "Default" in each of the above function names signifies that you can define alternative functions to replace these. If, for example, you define a function called TestcaseEnter, 4Test will invoke your function before executing any of the code in your test case and will not invoke `DefaultTestcaseEnter`.

`TestPlanEnter()` is not called until the first test case in the plan is run. Or the first marked test case, if you are only running marked test cases. Similarly, `TestPlanExit()` is called at the completion of the last marked test case. `TestPlanExit()` is only called if the last marked test description contains an executable test case, which means not a manual test case or a commented out test case specifier.

cs.inc

`cs.inc` is an automatically included file that contains functions used only in the multi-application environment. The following functions provide a recovery system for managing automated testing of client/server applications:

SetMultiAppStates	Sets an application state for each connected machine, if the "AppState" machine data lists one; if not, it calls the `DefaultBaseState` function, which sets the application to its main window.
SetMultiBaseStates	Sets the application to the lowest state in the application state hierarchy for each connected machine, if the "AppState" machine data lists an application state. The lowest application state is one in which the appstate declaration did not use the basedon keyword. If there is no "AppState" information associated with this machine, `SetMultiBaseStates` calls the `DefaultBaseState` function, which sets the application to its main window, invoking it beforehand if necessary.
SetUpMachine	Connects SilkTest to an Agent on the specified machine. It provides a way to associate a main window declaration and an application state function with a machine name. These parameters are stored as data accessible by means of the `GetMachineData` function. Both of these names (the second and third arguments to the function) are optional; however, if you omit both arguments, you will have no recovery system.
DefaultMultiTestCaseEnter	Executes at the beginning of a multi-test case. It invokes a `DisconnectAll` function. The invocation of the `SetAppState` function is performed by the `SetMultiAppStates` function because the `DefaultTestCaseEnter` function is not executed for a multi-test case.
DefaultMultiTestCaseExit	Executes just before a multi-test case terminates. It logs any pending exception, then invokes `SetMultiBaseStates` and `DisconnectAll`.

Include File Size

The maximum size of an include file is approximately 65536 lines. If your include file is very large, split it into two files and continue with your testing.

Troubleshooting

Handling Limited Licenses

By default, SilkTest starts up an unplanned Agent on the local workstation. If you do not want to use the local workstation as a test machine, set the Agent Name field in the **Runtime Options** dialog box to `(none)` instead of `(local)`. This will free up one license for a remote Agent.

Resolving Port-Number Conflicts

This functionality is available only for projects or scripts that use the Classic Agent. To configure port numbers for the Open Agent, see *Configuring SilkTest Open Agent Port Numbers*.

SilkTest connects to each Agent through a TCP/IP port that has a 4-digit or 5-digit ID. Typically, all the machines in your testing network automatically use the same default port number. This allows the

`Connect` function to automatically specify the port number for all connections. If some other application on one of your machines has already used the default port number, you have a port number conflict.

If you start an Agent on a PC and the default port is already in use, an error message is displayed.

In either case, you can use a different port number just for this machine, while using the default number for the rest, or you can have all your machines use the same available port number. When you have an Agent that uses a port number which is different than the default port number, you must specify the port number in every reference to that Agent. The syntax is `AgentName:nnnn` where `AgentName` is the target machine name and `nnnn` is the port number. Since you typically use a file or a list variable to hold your Agent names, you can add the `:nnnn` where needed.

If there are no port conflicts, you do not have to specify ports. If you have a conflict, the port number used for that machine must change. You can choose to change the port numbers used by all your PCs and workstations so that all use the same number.

Setting-Up Extensions for Distributed Testing

This functionality is available only for projects or scripts that use the Classic Agent.

If you are testing non-Web applications, you must disable browser extensions on your host machine. This is because the recovery system works differently when testing Web applications than when testing non-Web applications.

Furthermore, when you select one or both of the Internet Explorer (IE) extensions on the host machine's **Extension** dialog box, SilkTest automatically selects the correct version of the host machine's IE application in the **Runtime Options** dialog box. If the target machine's version of IE is not the same as the host machine's, you must remember to change the target machine's version.

Using Advanced Techniques

Starting from the Command Line

Starting SilkTest from the command line

You can start the SilkTest executable program from the command line by:

- Selecting Run from the Start menu.
- Using the command-line prompt in a DOS window or batch file.

The syntax is:

```
Partner [-complog filename] [-m mach] [-opt optionset.opt] [-p mess] [-proj
filename [-base filename]] [[-q] [-query query name] [-quiet] [-r filename] [-
resexport] [-resextract] [-r] scr.t/suite.s/plan.pln/link.lnk [args]] [-smlog
filename]
```

The `filename` specified for various options expects the file to be located in the working directory (the default location is the SilkTest install directory, `c:\Program Files\Silk\SilkTest\`). If you want to use a file that is located in another directory, you must specify the full path in addition to the filename.

Options

The following table lists all the options to the `partner` command.

args

Optional arguments to a script file. You can access the arguments using the `GetArgs` function and use them with your scripts.

If you pass arguments in the command line, the arguments provided in the command line are used and any arguments specified in the currently loaded options set are not used. To use the arguments in the currently loaded options set, do not specify arguments in the command line.

For more information, see *Passing arguments to a script*.

-complog

Tells SilkTest to log compilation errors to a file you specify. Enter the argument as `-complog filename`. For example: `partner [-complog c:\testing \logtest1.txt]`.

If you include this argument, each time you do a compilation SilkTest checks to see if the file you named exists. If it does not already exist, SilkTest creates and opens it. If the file already exists, SilkTest opens it and adds the information. The number of errors is written in the format (n error(s)) for example, 0 errors, 1 error, 50 errors. Compilation errors are written to the error log file as they appear in the "Errors" window. The error log file is automatically saved and closed when SilkTest finishes writing errors to it.

-m

Specifies the target machine. The default is the current machine. Call the 4Test built-in function `Connect` to connect to a different machine at runtime.

In order to use the `-m` switch, you need to have the **Runtime Options** dialog **Network setting** set to `TCP/IP` or `NetBIOS`. If this is set to `'(disabled)'`, the target machine is

ignored. To set the **Network setting**, either set it interactively in SilkTest **Runtime Options** before running from the command line, or save the setting in an option set and add the '-opt <option set>' argument to the command line.

-opt Specifies an options set. Must be followed by the path of the `.opt` file you want to use.

-p Provided for use with a Windows shell program that is running SilkTest as a batch task. The option enables another Windows program to receive a message containing the number of errors that resulted from the script(s) run. SilkTest broadcasts this message using the Windows PostMessage function, with the following arguments:

- `hWnd = HWND_BROADCAST`
- `uiMsg = RegisterWindowMessage (mess)`
- `wParam = 0`
- `lParam = number of errors`

To take advantage of the `-p` option, the shell program that runs SilkTest should first register mess, and should look for mess while SilkTest is running.

-proj Optional argument specifying the project file or archived project to load when starting SilkTest or SilkTest Runtime. For example, `partner –proj d:\temp\testapp.vtp –r agent.pln`.

`-base` is an optional argument to `–proj`. You use the base argument to specify the location where you want to unpack the package contents. For example, `partner –proj d:\temp\testapp.stp –base c:\rel30\testapp` unpacks the contents of the package to the `c:\rel30\testapp directory`.

-q Quits SilkTest after the script, suite, or testplan completes.

-query Specifies a query. Must be followed by the name of a saved query. Tells SilkTest to perform an **Include** > **Open All**, then **Testplan** > **Mark By Named Query**, then **Run** > **Marked Tests**.

-quiet Starts SilkTest in "quiet mode", which prevents any pop-up dialog boxes from appearing when SilkTest starts up.

The quiet option is particularly useful when used in conjunction with the `-smlog` option if you are doing unattended testing where a user is not available to respond to any pop-up dialog boxes that may appear. For example, `partner –quiet –smlog c:\testing \meter_dialog.txt` starts up SilkTest and saves any SilkMeter dialog boxes to the `meter_dialog.txt` file.

-r Must be the last option specified, followed only by the name of a SilkTest file to open. This includes files such as script (and, optionally, arguments that the script takes), a suite, testplan, or link file. If you specify a link file, tells SilkTest to resolve the link and attempt to open the link target. Otherwise, tells SilkTest to run the specified script, suite, or testplan, optionally passing args as arguments to a script file. For example, `partner –proj d:\temp\testapp.stp –base c:\rel30\testapp –r Agent.pln` unpacks the archive from the temp subdirectory into the `c:\rel30\testapp` subdirectory and then loads and executes the Agent.pln file.

-resexport Tells SilkTest to export a one line summary of the most recent results sets to `.rex` files automatically. Specifying `-resexport` has the same effect as if each script run invokes the `ResExportOnClose` function during its execution.

-resextract Tells SilkTest to extract all information from the most recent results sets to a `.txt` file. Both the SilkTest Extract menu command and the `-resextract` option create UTF-8 files.

script.t/ suite.s/ plan.pln/ link.lnk The name of the SilkTest script, suite, testplan, or link file to load, run, or open.

-smlog Tells SilkTest to log SilkMeter messages to a file you specify. The messages appear in the log file in addition to appearing in dialogs. Enter the argument as -smlog filename, for example: `partner [-smlog c:\testing\silkmeterlog.txt]`. Some, but not all SilkMeter messages cause SilkTest to close.

If you include the `-smlog` argument, SilkTest checks to see if the file you specify exists. If it does not already exist, SilkTest creates and opens it. If the file already exists, SilkTest opens it and appends the information. The SilkMeter log file is automatically saved and closed when SilkTest finishes writing errors to it.

See the example of using `-smlog` with the -quiet option.

Examples

To load SilkTest, type: `partner`

To run the test.s suite, type: `partner -r test.s on system "sys1"`

To run the test.t script, type: `partner -m sys1 -r test.t`

To run the test.t script with arguments, type: `partner -r test.t arg1 arg2`

To run the tests marked by the query named query3 in tests.pln, type: `partner -query query3 -r tests.pln`

To run tests.pln, and export the most recent results set from tests.res to tests.rex, type: `partner -q -resexport -r tests.pln`

To edit the test.inc include file, type: `partner test.inc`

Starting SilkTest Classic Agent from the command line

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can start the SilkTest Agent executable program from the command line by:

- Selecting **Start**>**Run**.
- Using the command-line prompt in a DOS window.

The syntax is:

```
agent [-p port]
```

Options

The following option is available for the `agent` command:

-p <port> Temporarily set network protocol to TCP/IP and port to specified <port>. For the Classic Agent, the default port is the value in the partner.ini file. If there is no value set in the partner.ini file, then the port is set to 2965. To permanently set a protocol and port, right-click the Agent window and select **Network**.

Port numbers may not be negative numbers or _D.

Examples

To load the SilkTest Classic Agent, enter:

```
agent
```

To load the SilkTest Classic Agent and specify network protocol to TCP/IP and port to 2965 (the default port for the Classic Agent), enter:

```
agent –p
```

To load the SilkTest Classic Agent and specify network protocol to TCP/IP and port to 1234, enter:

```
agent –p 1234
```

Recording a Test Frame

Overview of Classes

The `class` indicates the type, or kind, of GUI object being declared. Note that this is the 4Test class, not the class that the GUI itself uses internally. For example, although the class might be Label on one GUI and Text on another, 4Test uses the class name `StaticText` to refer to text strings that cannot be edited.

A class defines data and behavior

The class also defines methods (actions) and properties (data) that are inherited by the GUI object. For example, if you record a declaration for a pushbutton named OK, a testcase can legally use a method like `Click` on the pushbutton because the `Click` method is defined at the class level. In other words, the definition of what it means to click on a pushbutton is included within the definition of the 4Test class itself, and this definition is inherited by each pushbutton in the GUI. If this were not true, you would have to define within each GUI object's window declaration all the methods you wanted to use on that object.

The class as recorded cannot be changed

The one exception is that if the recorded class is `CustomWin` — meaning that SilkTest does not recognize the object — you can, when appropriate, map the class to one that is recognized, as described in Mapping custom classes to standard classes.

Custom classes

Enable an application to perform functions specific to the application and to enhance standard class functionality. Custom classes are also easy to maintain and can be extended easily by developers. All custom objects default to the built-in class, `CustomWin`.

Custom objects fall into two general categories:

Visible objects	Are those objects that SilkTest knows about, but cannot identify, for example, the icon in an About dialog. Two further categories of visible objects include: • Common objects are those that look and behave like standard objects, for example, a third-party object that looks and acts like a `PushButton`, but is recorded as a `CustomWin`. • Uncommon objects, on the other hand, have no relation to the existing standard objects. For example, an Icon. there is no corresponding `Icon` class.
Invisible objects	Those that SilkTest cannot recognize at all.

Overview of Object Files

Object files are the compiled versions of include (`.inc`) or script (`.t`) files. Object files are saved with an "o" at the end of the extension, for example, `.ino`, or `.to`. Object files cannot be edited; the only way to

change compiled objects is to recompile the include or script file. When you save a script or include file, a source file and an object file are saved. Object files are not platform-specific; you can use them on all platforms that SilkTest supports.

In order for SilkTest to run a script or include file that is in source form, it must compile it, which can be time-consuming. Object files, on the other hand, are ready to run. Note that you cannot call objects that exist in the object file (`.to`) from a testplan; you must have the script file (`.t`).

To disable saving object files during compilation, the **AutoComplete** options on the **General Options** dialog box as well as the **Save object files during compilation** option on the Runtime Options dialog box need to be unchecked.

SilkTest always uses object files if they are available. When you open a script file or an include file, SilkTest loads the corresponding object file as well, if there is one. If the object file is not older than the source file, SilkTest does not recompile the source file. The script is ready to run. If the source file is more recent, SilkTest recompiles the source file before the script is run. If you then later save the source file, SilkTest automatically saves a new object file.

If a file is loaded during compilation (that is, if you include a file in another file that is being compiled), SilkTest loads only the object file, if it exists and is newer than the corresponding source file.

Object files may not be backward-compatible, although sometimes they will be. Specifically, object files will not work with versions of SilkTest for which the list of GUI/browser types is different than for the version used to compile the object file. The list is in `4Test.inc`. (For example, object files created before 'mswxp' was added as the GUI type for Windows XP cannot be used with ST5.5 SP3, which includes the 'mswxp' GUI type.)

If you are using a `.ino` file, but during compilation SilkTest displays a message that the corresponding .inc file is missing, then you may be experiencing the object file version incompatibility explained in the preceding paragraph.

Advantages of object files

Advantages of object files include:

- Because object files are ready to run, they do not need to be recompiled if the source file has not changed. This can save you a lot of time. If your object file is more recent than your source file, the source file does not need to be recompiled each time the file is first opened in a session; the object file is used as is.
- You can distribute object files without having to distribute the source equivalents. So if you have built some tests (and include files) that you want to distribute but don’t want others to see the sources, you can distribute only the object files.

Since an object file cannot be run directly:

- Define the code you want to "hide" in an include file, which will be compiled into an `.ino` object file.
- Call those functions from an ordinary script file.
- Distribute the `.t` script file and the compiled `.ino` include file. Users can open and run the script file as usual, through **File** > **Run**.

Here’s a simple example of how you might distribute object files so that others cannot see the code.

In file `test.inc`, place the definition of a function called `TestFunction`. (When you save the file, the entire include file is compiled into `test.ino`.)

```
TestFunction ()
           ListPrint (Desktop.GetActive ())
```

In the file `test.t` use the `test.inc` include file. SilkTest will load the `.ino` equivalent. Call `TestFunction`, which was defined in the include file.

```
use "test.inc"
```

```
        main ()
        TestFunction () // call the function
```

Distribute `test.t` and `test.ino`. Users can open `test.t` and run it but do not have access to the actual routine, which resides only in compiled form in `test.ino`.

Declarations

Generic message box declaration

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

When SilkTest generates the window declarations for the main window of your application, it also includes a declaration for a generic object named MessageBox. Therefore, you do not have to record a declaration for each of the message boxes (potentially hundreds) in your application.

A message box is a dialog that has static text and pushbuttons, but no other controls. Typically, message boxes are used to prompt users to verify an action (such as "Save changes before closing?") or to alert users to an error.

The message box declaration is generic for three reasons:

- The dialog's tag specifies that its parent is the current active application.
- The most likely names for pushbuttons are accounted for: **OK**, **Cancel**, **Yes**, and **No**.
- The tag of the message is an index number, not the text of the message.

If your application contains message boxes that have extra pushbuttons or if your pushbuttons use different names, you need to add the declarations for those buttons to the declaration for the generic MessageBox object. For example, if a message box contains a Test pushbutton, you need to add the following lines to the recorded declaration:

```
PushButton Test
   tag "Test"
```

Here is the declaration for the generic message box:

```
window MessageBoxClass MessageBox
   tag "~ActiveApp/[DialogBox]$MessageBox"
   PushButton OK
      tag "OK"
   PushButton Cancel
      tag "Cancel"
   PushButton Yes
      tag "Yes"
   PushButton No
      tag "No"
   StaticText Message
      mswnt tag "#2"
      tag "#1"
```

GUI specifiers

Where SilkTest can detect a difference from one platform to the next, it automatically inserts a **GUI-specifier** in a window declaration to indicate the platform, for example `msw`.

For a complete list of the valid GUI specifiers, see *GUITYPE data type*.

Overview of Dialog Declarations

The declarations for the controls contained by a dialog are nested within the dialog's declaration to show the GUI hierarchy.

The declarations for menus are nested (indented) within the declaration for the main window, and the declarations for the menu items are nested within their respective menus. This nesting denotes the

hierarchical structure of the GUI, that is, the parent-child relationships between GUI objects. Although a dialog is not physically contained by the main window, as is true for menus, the dialog nevertheless logically belongs to the main window. Therefore, a parent statement within each dialog declaration is used to indicate that it belongs to the main window of the application.

In the sample Text Editor application, `MainWin` is the parent of the `File` menu. The `File` menu is considered a child of the `MainWin`. Similarly, all the menu items are child objects of their parent, the `File` menu. A child object belongs to its parent object, which means that it is either logically associated with the parent or physically contained by the parent.

Because child objects are nested within the declaration of their parent object, the declarations for the child objects do not need to begin with the reserved word window.

The following example from the Text Editor application shows the declarations for the **Find** dialog and its contained controls:

```
window DialogBox Find
   tag "Find"
   parent TextEditor
   StaticText FindWhatText
     multitag "Find What:"
       "$65535"
   TextField FindWhat
     multitag "Find What:"
       "$1152"
   CheckBox CaseSensitive
     multitag "Case sensitive"
       "$1041"
   StaticText DirectionText
     multitag "Direction"
       "$1072"
   RadioList Direction
     multitag "Direction"
       "$1056"
   PushButton FindNext
     multitag "Find Next"
       "$1"
   PushButton Cancel
     multitag "Cancel"
       "$2"
```

Overview of Window Declarations

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`. Because your testcases use logical names, if the object's actual name changes on the current GUI, on another GUI, or in a localized version of the application, you only need to change the tag in the window declarations. You do not need to change any of your scripts.

You can add variables, functions, methods, and properties to the basic window declarations recorded by SilkTest. For example, you can add variables to a dialog box declaration that specify what the tab sequence is, what the initial values are, and so on. You access the values of variables at runtime as you would a field in a record.

After you record window declarations for the GUI objects in your application and insert them into a declarations file, called an include file (`*.inc`), SilkTest references the declarations in the include file to identify the objects named in your test scripts. You tell SilkTest which include files to reference through the Use Files field in the Runtime Options dialog.

Main Window and Menu Declarations

The main window declaration

The main window declaration begins with the 4Test reserved word window. The term window is historical, borrowed from operating systems and window manager software, where every GUI object (for example main windows, dialogs, menu items, and controls) is implemented as a window.

As is true for all window declarations, the declaration for the main window is composed of a class, identifier, and tag. The following example shows the beginning of the default declaration for the main window of the Text Editor application:

```
window MainWin TextEditor
  multitag "Text Editor"
        "$C:\PROGRAMFILES\<SilkTest install directory>\SILKTEST\TEXTEDIT.EXE"
```

Part of Declaration	Value for TextEditor's main window.
Class	MainWin
Identifier	TextEditor
Tag	Two components in the multiple tag: • " Text Editor "—The application's caption • " executable path "—The window ID • " The full path of the executable file that invoked the application "

sCmdLine and wMainWindow constants

When you record the declaration for your application's main window and menus, the sCmdLine and wMainWindow constants are created. These constants allow your application to be started automatically when you run your testcases.

The sCmdLine constant specifies the path to your application's executable. The following example shows an sCmdLine constant for a Windows environment:

```
mswnt const sCmdLine =    "c:\program files\<SilkTest install directory>
\silktest\textedit.exe"
```

The `wMainWindow` constant specifies the 4Test identifier for the main window of your application. For example, here is the definition for the `wMainWindow` constant of the Text Editor application on all platforms:

```
const wMainWindow = TextEditor
```

Menu declarations

The following example from the Text Editor application shows the default main window declaration and a portion of the declarations for the File menu:

```
window MainWin TextEditor
  multitag "Text Editor"
    "$C:\PROGRAM FILES\<SilkTest install directory>\SILKTEST\TEXTEDIT.EXE"
   .
   .
   .
   Menu File
      tag "File"
      MenuItem New
         multitag "New"
           "$100"
```

Menus do not have window IDs, but menu items do, so by default menus are declared with the tag statement while menu items are declared with the `multitag` statement.

Window Declarations

Overview of Window Declarations

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`. Because your testcases use logical names, if the object's actual name changes on the current GUI, on another GUI, or in a localized version of the application, you only need to change the tag in the window declarations. You do not need to change any of your scripts.

You can add variables, functions, methods, and properties to the basic window declarations recorded by SilkTest. For example, you can add variables to a dialog box declaration that specify what the tab sequence is, what the initial values are, and so on. You access the values of variables at runtime as you would a field in a record.

After you record window declarations for the GUI objects in your application and insert them into a declarations file, called an include file (`*.inc`), SilkTest references the declarations in the include file to identify the objects named in your test scripts. You tell SilkTest which include files to reference through the Use Files field in the Runtime Options dialog.

Improving SilkTest Window Declarations

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The current methodology for identifying window declarations in Microsoft Windows-based applications during a recording session is usually successful. However, some applications may require an alternate approach of obtaining their declarations because their window objects are invisible to the SilkTest Recorder. You can try any of the following:

- Turning on Accessibility - use this if during a session started with the Recorder, SilkTest is unable to recognize objects within a Microsoft Windows-based application. This functionality is available only for projects or scripts that use the SilkTest Classic Agent.
- Defining a new window - use this if turning on Accessibility does not help SilkTest to recognize the objects. This functionality is available only for projects or scripts that use the SilkTest Classic Agent.
- Creating a testcase that uses dynamic object recognition - use this to create testcases that use XPath queries to find and identify objects. Dynamic object recognition uses a **Find** or **FindAll** method to identify an object in a testcase. This functionality is available only for projects or scripts that use the SilkTest Open Agent.

Improving SilkTest recognition by defining a new window

If SilkTest is having difficulty recognizing objects in Internet Explorer or Microsoft Office applications, try enabling Accessibility. If that does not help improve recognition, try defining a new window.

How defined windows works

When you use Defined Window, you use the mouse pointer to "draw a rectangle" around the object that SilkTest cannot record and then assign a name to the object. When you save your work, SilkTest stores the name and the object's coordinates in a test script. When you replay the script, SilkTest uses a `Click()` method on the center of the area you specified.

Notes

- Defining a new window is only available for projects or scripts that use the SilkTest Classic Agent.
- Defining a new window is not available for Java applications or applets.

- Defined Window does not support nesting of defined objects.
- Defined Window is location-based and uses pixel coordinates to locate the object in the parent window. Thus, if the layout of your parent window changes and/or the object's coordinates change frequently, you may need to re-define the window in order for SilkTest to correctly declare the object.
- If you draw a rectangle around an unrecognized object, but also include an object that SilkTest easily recognizes, SilkTest records both and lists the easily recognized object first.

Recording window declarations for the main window and menu hierarchy

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Start your application and in SilkTest, click **File** > **New**.
2. Click **Test Frame** and then click **OK**. SilkTest displays the **New Test Frame** dialog, which allows you to create a test frame file for an application displayed in the **Application** list box. By default, SilkTest names the new test frame file `frame.inc`, denoting it is an include file that contains declarations.
3. In the **File name** field, accept the default test frame name (`frame.inc`), or type a new name.

   If you change the default name of the file, make sure to include the file extension .inc in the new file name. If you do not, the file is not identified to SilkTest as an include file and SilkTest will give it a .txt extension and report a compilation error when you click OK to create the file.
4. Select your application from the **Application** list box. The Application list box displays all applications that are open and not minimized. If your test application is not listed, click **Cancel**, open your application, and choose **File** > **New** again.
5. Click **OK**. SilkTest creates the new test frame file. Window declarations appear in the testplan editor, which means that the declarations for individual GUI objects can be expanded to show detail, collapsed to hide detail, and edited if necessary.

Recording a window declaration for a dialog

After you record your test application's main window and menus, you record all the dialogs you want to test. Use this procedure once for each dialog in your application.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Make sure that the test frame (`.inc`) file that contains the declarations for the application's main window is open. The dialog declarations will be appended to this file.
2. Click **Record** > **Window Declarations**.
3. Make your application active and invoke one of its dialogs, referred to in this procedure as the target dialog. If necessary, arrange windows so that you can see the target dialog and position the cursor on the title bar of the target dialog.

   Note that as you move the cursor toward the title bar, the contents of the **Window Declaration** list box change dynamically to reflect the object at which you're pointing, as well as any contained objects. When the cursor is positioned correctly, the **Window Detail** group box (upper left) shows the caption of the dialog in the Identifier field.
4. Press `Ctrl+Alt`. The declaration is frozen in the lower half of the dialog.
5. Close the target dialog.
6. In the **Record Window Declarations** dialog, click **Paste to Editor**. The information in the **Record Window Declarations** dialog is cleared, and the newly recorded declarations are appended to the test frame after the last recorded declaration.
7. If you are finished recording declarations, click **Close** on the **Record Window Declarations** dialog. Otherwise, click **Resume Tracking** to begin recording the declarations for another dialog.

   Many applications begin with a login window, which is not accounted for when you record the test frame. Therefore, make sure that you invoke this window and record a declaration for it when you are recording the declarations for your application's dialogs.

Defining a new window

Defining a new window can improve how SilkTest records Microsoft Office-based and Internet Explorer applications.

You must have an include file open in order to define a new window.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Choose **Record** > **Defined Window**.
2. Click the **Draw Rectangle** button.
3. Click and drag to form a rectangle around the object you want SilkTest to record. Notice how the position of the rectangle is recorded in pixels in the **Window Rectangle** field.
4. Once you lift the mouse button, click **Add**.

   The Update Window Declaration Detail confirmation message containing a message similar to the one below:

   ```
   The window declarations in the following files will be updated c:\program
   files\<SilkTest installation directory>\silktest\projects\aaa\frame.inc.
   ```
5. Click **OK**. The information is saved to the specified file.

The file opens. Scroll to see the new `DefinedWin` with the name you assigned and a tag with the coordinates of the rectangle you drew.

```
[-] DefinedWin StartWindow
[ ] tag "(295,380-997,642)"
```

Specifying tags

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When you are recording declarations, you can select any combination of tags to record by selecting check boxes in the Tag Information group box in the **Record Window Declarations** dialog. You can record different tags for different objects. You can also specify which tags you want recorded by default.

Default tags

You can record more than one tag for an object. Doing so makes scripts less sensitive to changes when the tests are run. For example, if you record a caption and a window ID for a control, then even if the caption on the control changes (such as the caption "Case sensitive" changing to "Case is significant"), SilkTest can still find the control based on its window ID.

This is particularly an issue in situations where captions change dynamically, such as in MDI applications where the window title changes each time a different child window is made active.

By default, when you record window declarations, each object is given two tags: the caption (if there is one) and the Window ID (if there is one). Notice in the Record Declarations dialog shown above that two tags are checked in the Tag Information box: Caption and Window ID.

For example, here is the default recorded declaration for the Case sensitive check box:

```
CheckBox CaseSensitive
   multitag "Case sensitive"
            "$1041"
```

SilkTest specifies multiple tags in a declarations file using the multitag statement. In the previous example, the check box is declared with two tags:

- The string "Case sensitive", which is its caption.
- The string "$1041", which is its Window ID.

Using class-specific multiple tags

You can specify which multiple-tag types to use for an individual class. For example, maybe you don't want window ID used with a particular class, even though you want window ID used with all other classes. You can specify this by including a setting statement in the declaration for the class.

For more information, see the `winclass declaration`.

Multiple tags at runtime

When running your testcases, the Agent tries to resolve each part of a multiple tag from top to bottom until it finds an object that matches.

Consider this declaration:

```
CheckBox CaseSensitive
   multitag "Case sensitive"
            "#1"
```

When SilkTest encounters a reference to `Find.CaseSensitive`, it first looks for a check box whose caption is "Case sensitive". If it finds one, it uses it. If it doesn't find one, it looks for the first check box in the dialog (because of the index tag "#1"). If there is one, SilkTest uses it. If none of the tags resolve, an exception is raised.

For complete information about tag resolution, see `multitag` statement.

Changing tags

Sometimes you need to change tags from what SilkTest named them by default. See *Changing the tags recorded by default*.

Why change the tags

By default, the GUI object's caption and index are used for the tag, because they are the most portable. In most cases, these are what you want to use.

However, there are situations in which the default tag is not suitable. For complete information about using alternative tags, see the section *How to choose a tag-string form in multitag statement*.

Example: changing a tag

You might want to provide more than one caption for a control if the control's caption can change dynamically. For example, if a push button sometimes says Yes and sometimes says Continue, you could change the tag as shown here:

```
PushButton Confirm
           multitag "Yes"
  "Continue"
```

The Agent would find the pushbutton if it had either caption.

To separate different tag components, use the pipe character: `|`.

Modify a declaration in the Record Window Declarations dialog

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can modify the identifier or tag for a dialog as you record it.

1. In the **Window declaration** list box at the bottom of the dialog, click the line for the object containing the tag or identifier you want to change. SilkTest updates the **Window Detail** group box in the upper left of the dialog to include the information from the line you clicked.

2. To change the identifier, replace the existing identifier with one of your choice.
3. To change the tag, select the tag types you want to include in the generated `multitag`. You can edit the contents of each tag type in the text fields in the **Tag Information** group box. The **Window declaration** list box updates dynamically as you enter the new information.
4. When finished making modifications, click **Paste to Editor**. SilkTest appends the declarations for the dialog to the test frame file.

Changing the tags recorded by default

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Click **Record** > **Window Declarations**.
2. Click **Options**.
3. On the **Record Window Declarations Options** dialog, check and uncheck the check boxes in the **Default multitags** box as appropriate.
4. Click **OK**. The next time you record window declarations, SilkTest will use the tag types you selected by default. You can always override the defaults for a particular object.

Turning off multiple tag recording

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Click **Record** > **Window Declarations**.
2. Click **Options**. SilkTest displays the **Record Window Declarations Options** dialog.
3. Uncheck the **Record multiple tags** check box. The check boxes in the **Default tag** group box become option buttons.
4. Select the tag type you want SilkTest to use by default.
5. Click **OK**.

When you record window declarations, SilkTest defaults to the tag type you selected and record the tag in a `tag statement`. You can always override the default for a particular object.

For more information, see the *tag statement*.

Use the member-of operator to access data

Use the member-of operator (`.`) to reference the data defined in a window declaration. For example, if a script needs to know which control should have focus when the **Find** dialog is first displayed, it can access this data from the window declaration with this expression:

```
Find.lwTabOrder[1]
```

Similarly, to set focus to the third control in the list:

```
Find.lwTabOrder[3].SetFocus ()
```

Identifiers and Tags

Captions for objects

By default, SilkTest follows these steps to create a **Caption** tag for an object:

1. SilkTest uses the literal label or caption of the object, if there is one.
2. If the object has a sibling object with the same label or caption, SilkTest appends the object's index number to the tag. The index number is the object's order of appearance in relation to other sibling objects of the same class, from top left to bottom right within the parent object.

For example, if a dialog has two objects labeled **Find**, the tag of the one nearest the top left of the dialog is **Find[1]** and the tag of the one nearest the bottom right of the dialog is **Find[2]**.

3. If the object does not have a label or caption, SilkTest uses the index number.

   For example, if a dialog contains two unnamed text fields, the text field closest to the upper left corner of the dialog has the tag **#1**, and the other has the tag **#2**.

Overview of Identifiers

When you record testcases, SilkTest uses the window declarations in the test frame file to construct a unique identifier, called a fully qualified identifier, for each GUI object. The fully-qualified identifier consists of the identifier of the object, combined with the identifiers of the object's ancestors. In this way, the 4Test commands that are recorded can manipulate the correct object when you run your testcases.

If all identifiers were unique, this would not be necessary. However, because it is possible to have many GUI objects with the same identifier (for example, the **OK** button), a method call must specify as many of the object's ancestors as are required to uniquely identify it.

The following table shows how fully qualified identifiers are constructed:

GUI Object	Fully-Qualified Identifier	Example
Main Window	The main window's identifier	`TextEdit.SetActive ()`
Dialog	The dialog's identifier	`Find.SetActive ()`
Control	The identifiers of the dialog and the control	`Find.Cancel.Click ()`
Menu item	The identifiers of the main window, the menu, and the menu item	`TextEditor.File.Open.Pick ()`

The fully qualified identifier for main windows and dialogs does not need to include ancestors because the declarations begin with the keyword window.

An identifier is the GUI object's logical name. By default, SilkTest derives the identifier from the object's actual label or caption, removing any embedded spaces or special characters (such as accelerators). So, for example, the **Save As** label becomes the identifier **SaveAs**. Identifiers can contain single-byte international characters, such as é and ñ.

If the object does not have a label or caption, SilkTest constructs an identifier by combining the class of the object with the object's index. The index is the object's order of appearance, from top left to bottom right, in relation to its sibling objects of the same class. For example, if a text field does not have a label or caption, and it is the first text field within its parent object, the default identifier is TextField1.

Note that the identifier is arbitrary, and you can change the generated one to the unique name of your choice.

Overview of Tags

The tag is the actual name of the object, as opposed to the identifier, which is the logical name. SilkTest uses the tag to identify objects in the application under test when recording and when executing testcases. Testcases never use the tag to refer to an object; they always use the identifier.

Alternatively, 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.

There are several types of tags:

Tag Type	Description
`Caption`	The caption or label as it appears to the user.

Tag Type	Description
`Prior text`	Closest static text above or to the left of the object. Prior text tags begin with the ^ character.
`Index`	The order (from top left to bottom right) in relation to its sibling objects of the same class. Index tags must begin with the # character.
`Window ID`	The GUI-specific internal ID of the object. Window ID tags begin with the $ character.
`Location`	The physical location (coordinates) of the object. Location tags begin with the @ character.
`Attributes`	The attribute name(s) of the Html object. If the object is not an Html object, nothing is recorded.

Not all types of objects have all tags. Dialogs, for example, do not have window IDs, so they cannot have a `Window ID` tag.

In the **Record Window Declarations** dialog, if you record declarations for the **Case sensitive** check box in the **Text Editor's Find** dialog, the possible tags for the check box include:

Tag Type	Value	Comments
`Caption`	Case sensitive	
`Prior text`	^Find What:	"Find What" is the nearest static text above or to the left of the check box
`Index`	#1	The Case Sensitive check box is the first check box in the dialog
`Window ID`	$1041	
`Location`	@(57,75)	
`Attributes`	[blank]	Attributes are only recorded for Html objects.

These are the possible tags that can be used by SilkTest to identify the Case sensitive check box when recording or executing testcases.

It is helpful to understand how SilkTest identifies tags in browsers; for more information, see *Comparison of DOM and VO*.

Accessibility

Enabling Accessibility

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you are testing an application and SilkTest cannot recognize objects, you should first enable accessibility. Accessibility is designed to help SilkTest recognize objects at the class level. If that does not help with recognition, then you should try defining a new window by clicking **Record** > **Defined Window**.

Accessibility is turned on by default, but you need to enable your extension as usual. There are two ways to enable accessibility:

- If you are using the **Basic Workflow**, SilkTest is usually able to do this automatically if you check the **Enable Accessibility** check box on the **Extension Settings** dialog.
- If you are configuring your extension manually, you can enable Accessibility by checking the **Accessibility** check box on the **Extension Enabler** and the **Extensions Option** dialog.

Adding Accessibility classes

Use accessibility to help SilkTest better identify unrecognizable objects. The information you add to the list of classes is stored in the `accex.inc` file, which is installed by default in the `<SilkTest installation directory>\Extend` or `<name of project>\extend` directory.

You cannot add duplicate or blank class names.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Open the application containing the unrecognizable objects.
2. Open SilkTest, then choose **Record** > **Class** > **Accessibility**.
3. On the Windows Accessibility dialog, click and drag the **Finder tool** icon over the object you want to identify. A black rectangle appears around the edge of the control. When you release the mouse button, the object's information appears in the **Name** and **Class** fields.
4. Click **Add** to move the class name to the list of Accessibility classes that SilkTest can identify, then click **OK**.

Improving SilkTest object recognition with Accessibility

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

There are several objects in Internet Explorer and Microsoft applications that SilkTest can better recognize if you enable accessibility. For example, without enabling accessibility SilkTest records only basic information about the menu bar in Microsoft Word and the tabs that appear in Internet Explorer 7.0. However, with accessibility enabled, SilkTest fully recognizes those objects.

Accessibility is not available for Java applications or applets.

Comparison of what SilkTest records

Without Accessibility enabled on Microsoft Excel, SilkTest records the following if the mouse points to the File command on the main toolbar:

```
[+] CustomWin MenuBar
[+] multitag "[MsoCommandBar]Menu Bar"
[ ] "[MsoCommandBar]$0[1]"
[+] CustomWin TypeAQuestionForHelp
[+] multitag "[RichEdit20W]Type a question for help"
[ ] "[RichEdit20W]$16075636"
[+] CustomWin Standard
[+] multitag "[MsoCommandBar]Standard"
[ ] "[MsoCommandBar]$0[2]"
...
```

However, if you record the same testcase with accessibility enabled, SilkTest is able to record the following:

```
[+] AccObject WorksheetMenuBar
    [+] multitag "Worksheet Menu Bar"
        [ ] "$window"
[+] AccObject WorksheetMenuBar2
    [+] multitag "Worksheet Menu Bar[2]"
     [ ] "$menu bar[2]"
    [+] AccMenuItem File
        [+] multitag "File"
            [ ] "$menu item[1]"
    [+] AccMenuItem Edit
        [+] multitag "Edit"
         [ ] "$menu item[2]"
    [+] AccMenuItem View
        [+] multitag "View"
            [ ] "$menu item[3]"
```

```
    [+] AccMenuItem Insert
        [+] multitag "Insert"
            [ ] "$menu item[4]"
    [+] AccMenuItem Format
        [+] multitag "Format"
            [ ] "$menu item[5]"
…
```

With accessibility enabled SilkTest is able to record more than simple details about the File menu command.

SilkTest stores the information about accessibility classes in the accex.inc file which is installed by default in the `<SilkTest installation directory>/Extend or <name of project>\extend` directory. The `accex.inc` file comes preloaded with several classes, including the MsoCommandBar, the class of the Microsoft Office menu bar.

Removing Accessibility classes

This applies to the SilkTest Classic Agent, not for the SilkTest Open Agent.

You use accessibility to help better identify unrecognizable objects. However, you may want to delete classes from the list of classes you created in order to clean up your `.inc` file or to prevent recognition of classes.

1. Choose **Record** > **Class** > **Accessibility**.
2. On the **Windows Accessibility** dialog, select the name of the class you want to remove from the list of Accessibility classes.
3. Click **Remove** to delete the class name, then click **OK**.

The information is removed from the list of classes in the `accex.inc` file. The `accex.inc` file is installed by default to the `<SilkTest installation directory>\Extend` or `<name of project>\extend` directory.

Recording a method for a GUI Object

If you need to perform an action on an object, and the class does not provide a method for doing so, you can define your own method in the window declaration for the object. Then, in your scripts, you can use the method as though it were just another of the class's built-in methods. You can hand-code methods or record them.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Before you can record a method, you must have already recorded window declarations.

1. Position the insertion point on the declaration of the GUI object to which you want to add a method.
2. Click **Record** > **Method**.
3. On the **Record Method** dialog, name the method by typing the name or selecting one of the predefined methods: `BaseState`, `Close`, `Invoke`, or `Dismiss`.
4. Click **Start Recording**. SilkTest is minimized and your application and the SilkTest Record Status dialog open.
5. Perform and record the actions that you require.
6. On the **SilkTest Record Status** dialog, click **Done**. The **Record Method** dialog opens with the actions you recorded translated into 4Test statements.
7. On the Record Method dialog, click **OK** to paste the code into your include file.
8. Edit the 4Test statements that were recorded, if necessary.

Example: defining a method for a GUI object.

For example, suppose you want to create a method named `SetLineNum` for a dialog named **GotoLine**, which performs the following actions:

- Invokes the dialog
- Enters a line number
- Clicks the **OK** button

The following 4Test code shows how to add the definition for the SetLineNum method to the GotoLine dialog's declaration:

```
window DialogBox GotoLine
   tag "Goto Line"
   parent TextEditor
   const wInvoke = TextEditor.Search.GotoLine

   void SetLineNum (STRING sLine)
     Invoke ()            // open dialog
     Line.SetText (sLine) // populate text field
                          // whose identifier is Line
     Accept ()            // close dialog, accept values

     //Then, to go to line 7 in the dialog, you use this method call in your
testcases:
     GotoLine.SetLineNum (7)
```

Save the testframe

To save a test frame, click **File** > **Save** when the test frame is the active window. If it is a new file, it is automatically named `frame.inc` (if you already have a `frame.inc` file, a number is appended to the file name). You can use **File** > **Save** to select another name.

If you are working within a project, SilkTest automatically adds the new test frame (`.inc`) to the project.

When saving a file, SilkTest does the following:

- Saves a source file, giving it the `.inc` extension. The source file is an ASCII text file, which you can edit. For example: `myframe.inc`.
- Saves an object file, giving it the `.ino` extension. The object file is a binary file that is executable, but not readable by you. For example: `myframe.ino`.

Specifying how a dialog is invoked

4Test provides two equivalent ways to invoke a dialog:

- Use the `Pick` method to pick the menu item that invokes the dialog. For example: `TextEditor.File.Open.Pick ()`
- Use the Invoke method: `Open.Invoke ()`

While both are equivalent, using the `Invoke` method makes your testcases more maintainable. For example, if the menu pick changes, you only have to change it in your window declarations, not in any of your testcases.

The invoke method

To use the Invoke method, you should specify the dialog's winvoke variable, which contains the identifier of the menu item or button that invokes the dialog. For example:

```
window DialogBox Open
   tag "Open"
   parent TextEditor
WINDOW wInvoke = TextEditor.File.Open
```

Class Attributes

Attributes tag notation

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The attributes tag does not use the [n] notation to distinguish between windows with the same value of the tag. If the attributes tag is comprised only of an attribute that is not unique, then the same tag is recorded for multiple objects. The duplication is not detected until SilkTest tries to match the tag at runtime, at which time an `E_WINDOW_NOT_UNIQUE` exception will be raised.

If you anticipate that multiple windows may have the same value for the attributes tag, then use `multitags` instead (the attributes tag will have precedence).

Enabling class attribute recording

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You must turn on attribute recording in order for SilkTest to capture Html class attributes.

1. Click **Record** > **Window Declarations**, then click **Options**.
2. On the **Record Window Declarations Options** dialog, check the **Attributes** check box in the Default multi-tag area.

   As with other attributes, if you want to record only the Attributes tags, uncheck the **Record multiple tags** check box. As with previous versions of SilkTest, if you want to record multiple tags, be sure to leave the Record multiple tags check box checked.
3. Click **OK** to save your selection.

Recording existing Html class attributes and specifying the hierarchy of attributes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You must first Enable attribute recording in order to have SilkTest capture Html class attribute information while recording.

This task is supported on the SilkTest Classic *Agent* only.

1. Click **Record** > **Window Declarations**, then click the **Edit Class Attributes** button.
2. On the **Edit Class Attributes** dialog, select **Browser DOM** from the **Set** list box.
3. Select a class from the **Class** list box. For example, select **HtmlCheckBox** to specify attributes within the `HtmlCheckBox` class that you want to record.
4. After you have selected a class, select the attribute you want to record from the list of attributes in the **Defined attributes** list box.
5. Click the **>>** button to move the attribute to the **Class attributes** pane. You can only select a single attribute at a time.
6. Select an attribute, then click the **Move Up** or **Move Down** button to indicate the order in which you want the attributes to appear when you paste the window declaration to the Editor. You may select only a single attribute at a time.
7. Click **OK** to save your work and return to the **Record Window Declarations**. SilkTest will record the attribute tag(s) in the order you specified.

You can record custom class attributes in a test that uses hierarchical object recognition and SilkTest will record the attribute tags in the order you specified.

For example, you decide to record four attributes for the HtmlImage class. First you select id from the Defined Attributes list, then you click the **>>** button to move it to the **Class Attributes** list. You repeat that

process for the name, rel, and src attributes. Once the four attributes are listed in the **Class Attributes** list, you use the **Move Up** and **Move Down** buttons so that they appear in the order you want the attributes to appear.

After you click **OK**, SilkTest displays the **Record Window Declarations** dialog. Whenever SilkTest recognizes an `HtmlImage` object, SilkTest records the attributes you specified on the **Edit Class Attributes** dialog. The full string for the attributes tag information that SilkTest records is:

```
&id='myButton';name='button';src='file:???D:?buttonnext.gif';rel='start'
```

Adding a new class attribute and specifying the hierarchy of attributes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can add a new attribute to the list that SilkTest records for an `Html` class and specify the order in which the attributes are recorded. Add custom attributes to a Web application to make a test more specific.

To capture class attribute information, you must first Enable attribute recording.

1. Click **Record** > **Window Declarations**, then click the **Edit Class Attributes** button.
2. On the **Edit Class Attributes** dialog, select **Browser DOM** from the **Set** list box.
3. Select a class from the **Class** list box. For example, select `HtmlCheckBox` to specify attributes within the `HtmlCheckBox` class that you want to record.
4. Type the name of the new attribute in the field above the **Add** and **Remove** buttons, then click **Add**.

   There is a 62 character limit to attribute names. You may type in the name or you can copy and paste the name from the text editor. If you are typing in a long name, the field stops accepting characters after the 62nd character. If you paste in a long name, the name is truncated at 62 characters. The following characters are not allowed in attribute names: `(space) ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/`
5. Click the **>>** button to move the new attribute to the **Class Attributes** pane.
6. Select an attribute, then click the **Move Up** or **Move Down** buttons to indicate the order in which you want the attributes to appear when you paste the window declaration to the Editor. You may select only a single attribute at a time.
7. Click **OK** to save your work and return to the Record Window Declarations dialog. SilkTest will now record the new attribute tags in the order you specified.

   To record a test that uses the custom `Html` class attribute, you must use the Classic Agent.

Deleting a class attribute

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can delete an Html class attribute or a Java SWT custom class attribute if you want SilkTest to avoid recording it.

1. Click **Record** > **Window Declarations**, then click the **Edit Class Attributes** button.
2. On the **Edit Class Attributes** dialog, select the name of the attribute you want to delete from the **Defined attributes** list box.
3. Click **Remove**. The name is removed from the list of attributes. It is possible to delete an attribute from the Defined attributes list and still have it appear in the Class attributes list. An attribute remains in the Class attributes list until you move it to the Defined attributes list and then (if you like) delete it.
4. Click **OK** to save your work and return to the **Record Window Declarations** dialog, or click **Cancel** to avoid deleting the attribute.

Calling Windows DLLs from 4Test

Aliasing a DLL name

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If a DLL function has the same name as a 4Test reserved word, or the function does not have a name but an ordinal number, you need to rename the function within your 4Test declaration and use the 4Test alias statement to map the declared name to the actual name.

For example, the exit statement is reserved by the 4Test compiler. Therefore, to call a function named exit, you need to declare it with another name, and add an alias statement, as shown here:

```
dll "mydll.dll"
my_exit ()
alias exit
```

Calling a DLL from within a 4Test script

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

A declaration for a DLL begins with the keyword dll. The general format is:

```
dll dllname.dll
prototype
[prototype]...
```

where `dllname` is the name of the dll file that contains the functions you want to call from your 4Test scripts and `prototype` is a function prototype of a DLL function you want to call.

Prototype syntax

A function prototype has this form: `return-type func-name ( [arg-list] )`

where:

`return-type`: the data type of the return value, if there is one.

`func-name`: an identifier that specifies the name of the function.

`arg-list`: a list of the arguments passed to the function, specified as follows:

`[pass-mode] data-type identifier`

`pass-mode`: Specifies whether the argument is passed into the function (`in`), passed out of the function (`out`), or both (`inout`). If omitted, `in` is the default.

To pass by value, make a function parameter an `in` parameter.

To pass by reference, use an `out` parameter if you only want to set the parameter's value; use an `inout` parameter if you want to get the parameter's value and have the function change the value and pass the new value out.

`data-type`: the data type of the argument.

`identifier`: the name of the argument.

You can call DLL functions from 4Test scripts, but you cannot call member functions in a DLL.

Passing arguments to DLL functions

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

Valid data types for arguments passed to DLL functions

Since DLL functions are written in C, the arguments you pass to these functions must have the appropriate C data types. In addition to the standard 4Test data types, SilkTest also supports these ten C data types:

- char, int, short, and long
- unsigned char, unsigned int, unsigned short, and unsigned long
- float and double
- Any argument you pass must have one of these data types (or be a record that contains fields of these types).

Passing string arguments

The `char*` data type in C is represented by the 4Test `STRING` data type. The default string size is 256 bytes.

The following code fragments show how a char array declared in a C struct is declared as a `STRING` variable in a 4Test record:

```
// C declaration
typedef struct
{
...
char szName[32];
...
}

// 4Test declaration
type REC is record
...
STRING sName, size=32
...
```

To pass a NULL pointer to a STRING, use the NULL keyword in 4Test. If a DLL sets an out parameter of type char* to a value larger than 256 bytes, you need to initialize it in your 4Test script before you pass it to the DLL function. This will guarantee that the DLL does not corrupt memory when it writes to the parameter. For example, to initialize an out parameter named `my_parameter`, include the following line of 4Test code before you pass my_parameter to a DLL: `my_parameter = space(1000)`

If the user calls a DLL function with an output string buffer that is less then the minimum size of 256 characters, the original string buffer is resized to 256 characters and a warning is printed. This warning, `String buffer size was increased from x to 256 characters` (where x is the length of the given string plus one) alerts the user to a potential problem where the buffer used might be shorter than necessary.

Passing arguments to functions that expect pointers

When passing pointers to C functions, use these conventions:

- Pass a 4Test string variable to a DLL that requires a pointer to a character (null terminated).
- Pass a 4Test array or list of the appropriate type to a DLL that requires a pointer to a numerical array.
- Pass a 4Test record to a DLL that requires a pointer to a record. 4Test records are always passed by reference to a DLL.
- You cannot pass a pointer to a function to a DLL function.

Passing arguments that can be modified by the DLL function

An argument whose value will be modified by a DLL function needs to be declared using the `out` keyword. If an argument is sometimes modified and sometimes not modified, then declare the argument as `in` and then, in the actual call to the DLL, preface the argument with the `out` keyword, enclosed in brackets.

For example, the third argument (`lParam`) to the `SendMessage` DLL function can be either `in` or `out`. Therefore, it is declared as follows:

```
// the lParam argument is by default an in argument
dll "user.dll"
LRESULT
SendMessage (HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)
```

Then, to call the DLL with an out argument, you use the keyword `out`, enclosed within brackets:

```
SendMessage (Open.hWnd, WM_GETTEXT, 256, [out] sText)
```

For more information, see dll declaration.

Passing window handles to a DLL function

If a parameter takes a window handle, use the `hwnd` property or the `GetHandle` method of the `AnyWin` class to get the window handle you need.

Using DLL support files installed with SilkTest

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest is installed with the following include files that contain all the declarations, data types, and constants necessary for you to call hundreds of functions within the Windows API from your scripts.

`msw32.inc` Contains use statements for the include files that apply to 32-bit Windows: `mswconst.inc`, `mswtype.inc`, `mswfun32.inc`, `mswmsg32.inc`, and `mswutil.inc`.

By including `msw32.inc` in your 4Test scripts, you have access to all the information in the other include files.

Note that the DLL functions declared in the files included in `msw32.inc` are aliased to the W (wide-character) functions.

`mswconst.inc` Declares constants you pass to DLL functions. These constants contain style bits, message box flags, codes used by the `GetSystemMetrics` function, flags used by the `GetWindow` function, window field offsets for the `GetWindowLong` and the `GetWindowWord` functions, class field offsets for the `GetClassLong` and `GetClassWord` functions, and menu function flags.

`mswfun32.inc` Contains 4Test declarations for 32-bit functions in the `user32.dll` and `kernel32.dll` files. The `mswfun32.inc` file provides wide character support. This means that you no longer have to edit `mswfun32.inc` in order to call Windows DLL functions. See the description of `mswfun32.inc` in the Dll declaration section.

`mswmsg32.inc` Declares 32-bit Microsoft Window messages, control messages, and notification codes.

`mswtype.inc` Declares many data types commonly used in the Windows API.

`mswutil.inc` Contains the following utility functions:

- `PrintWindowDetail`
- `GetStyleBitList`
- `PrintStyleBits`

Extending the Class Hierarchy

Classes

Overview of Classes

The `class` indicates the type, or kind, of GUI object being declared. Note that this is the 4Test class, not the class that the GUI itself uses internally. For example, although the class might be Label on one GUI and Text on another, 4Test uses the class name `StaticText` to refer to text strings that cannot be edited.

A class defines data and behavior

The class also defines methods (actions) and properties (data) that are inherited by the GUI object. For example, if you record a declaration for a pushbutton named OK, a testcase can legally use a method like `Click` on the pushbutton because the `Click` method is defined at the class level. In other words, the definition of what it means to click on a pushbutton is included within the definition of the 4Test class itself, and this definition is inherited by each pushbutton in the GUI. If this were not true, you would have to define within each GUI object's window declaration all the methods you wanted to use on that object.

The class as recorded cannot be changed

The one exception is that if the recorded class is `CustomWin` — meaning that SilkTest does not recognize the object — you can, when appropriate, map the class to one that is recognized, as described in Mapping custom classes to standard classes.

Custom classes

Enable an application to perform functions specific to the application and to enhance standard class functionality. Custom classes are also easy to maintain and can be extended easily by developers. All custom objects default to the built-in class, `CustomWin`.

Custom objects fall into two general categories:

Visible objects	Are those objects that SilkTest knows about, but cannot identify, for example, the icon in an About dialog. Two further categories of visible objects include: • Common objects are those that look and behave like standard objects, for example, a third-party object that looks and acts like a `PushButton`, but is recorded as a `CustomWin`. • Uncommon objects, on the other hand, have no relation to the existing standard objects. For example, an Icon. there is no corresponding `Icon` class.
Invisible objects	Those that SilkTest cannot recognize at all.

A Polymorphism

If a class defines its own version of a method or property, that method or property overrides the one inherited from an ancestor. This is referred to as polymorphism. For example, the `ListBox` class has its own `GetContents` method, which overrides the `GetContents` method inherited from the `AnyWin` class.

CursorClass ClipboardClass and AgentClass

There are three classes that are not part of the `AnyWin` class hierarchy, because they define methods for objects that are not windows:

- `CursorClass`, which defines the three methods you can use on the cursor: `GetPosition`, `GetType`, and `Wait`.

- `ClipboardClass`, which defines the two methods you can use on the system clipboard: `GetText` and `SetText`.
- `AgentClass`, which defines the methods you can use to set options in the 4Test Agent. (The 4Test Agent is the component of SilkTest that translates the method calls in your testcases into the appropriate GUI- specific event streams.)

Predefined identifiers for Cursor, Clipboard, and Agent

You do not record declarations for the cursor, the clipboard, or the Agent. Instead, you use predefined identifiers for each of these objects when you want to use a method to act against the object. The predefined methods for each are:

- 4Test Agent: `Agent`
- clipboard: `Clipboard`
- cursor (mouse pointer): `Cursor`

For example, to set a 4Test Agent option, you use a call such as the following:

```
Agent.SetOption (OPT_VERIFY_COORD, TRUE)
```

Defining new classes

Consider the declarations for the **Open** and the **Save As** dialogs of the **Text Editor** application, which each contain exactly the same child windows:

window DialogBox Open

```
   tag "Open"
   parent TextEditor
   StaticText FileNameText
      tag "File Name:"
   TextField FileName1
      tag "File Name:"
   ListBox FileName2
      tag "File Name:"
   StaticText DirectoriesText
      tag "Directories:"
   StaticText PathText
      tag "#3"
   ListBox Path
      tag "#2"
   StaticText ListFilesOfTypeText
      tag "List Files of Type:"
   PopupList ListFilesOfType
      tag "List Files of Type:"
   StaticText DrivesText
      tag "Drives:"
   PopupList Drives
      tag "Drives:"
   PushButton OK
      tag "OK"
   PushButton Cancel
      tag "Cancel"
   PushButton Network
      tag "Network"
```

window DialogBox SaveAs

```
   tag "Save As"
   parent TextEditor
   StaticText FileNameText
      tag "File Name:"
   TextField FileName1
```

```
      tag "File Name:"
   ListBox FileName2
      tag "File Name:"
   StaticText DirectoriesText
      tag "Directories:"
   StaticText PathText
      tag "#3"
   ListBox Path
      tag "#2"
   StaticText ListFilesOfTypeText
      tag "List Files of Type:"
   PopupList ListFilesOfType
      tag "List Files of Type:"
   StaticText DrivesText
      tag "Drives:"
   PopupList Drives
      tag "Drives:"
   PushButton OK
      tag "OK"
   PushButton Cancel
      tag "Cancel"
   PushButton Network
      tag "Network"
```

It is not uncommon for an application to have multiple dialogs whose only difference is the caption: The child windows are all identical or nearly identical. Rather than recording declarations that repeat the same child objects, it is cleaner to create a new class that groups the common child objects.

For example, here is the class declaration for a new class called `FileDialog`, which is derived from the `DialogBox` class and declares each of the children that will be inherited by the **SaveAs** and **Open** dialogs:

```
winclass FileDialog : DialogBox
   parent TextEditor
   StaticText FileNameText
      tag "File Name:"
   TextField FileName1
      tag "File Name:"
   ListBox FileName2
      tag "File Name:"
   StaticText DirectoriesText
      tag "Directories:"
   StaticText PathText
      tag "#3"
   ListBox Path
      tag "#2"
   StaticText ListFilesOfTypeText
      tag "List Files of Type:"
   PopupList ListFilesOfType
      tag "List Files of Type:"
   StaticText DrivesText
      tag "Drives:"
   PopupList Drives
      tag "Drives:"
   PushButton OK
      tag "OK"
   PushButton Cancel
      tag "Cancel"
   PushButton Network
      tag "Network"
```

To make use of this new class, you must do the following:

1. Rewrite the declarations for the **Open** and **Save As** dialogs, changing the class to **FileDialog**.
2. Remove the declarations for the child objects inherited from the new class.

Here are the rewritten declarations for the **Open** and **Save As** dialogs:

```
window FileDialog SaveAs
   tag "Save As"
window FileDialog Open
   tag "Open"
```

For more information on the syntax used in declaring new classes, see the winclass declaration.

The default behavior of SilkTest is to tag all instances of the parent class as the new class. So, if you record a window declaration against a standard object from which you have defined a new class, SilkTest records that standard object's class as the new class. To have all instances declared by default as the original class, add the following statement to the declaration of your new class: `setting DontInheritClassTag = TRUE`. For example, let's say you define a new class called `FileDialog` and derive it from the `DialogBox` class. Then you record a window declaration against a dialog box. SilkTest records the dialog box to be of the new `FileDialog` class, instead of the `DialogBox` class. To have SilkTest declare the dialog box's class as `DialogBox`, in the FileDialog definition, set DontInheritClassTag to TRUE. For example:

```
[+] winclass FileDialog : DialogBox
[ ] setting DontInheritClassTag = TRUE
```

Defining new class properties

You can define new properties for existing classes using the `property` declaration. You use these class properties to hold data about an object; you can use class properties anywhere in a script.

For complete information, see *property declaration*.

DesktopWin

Because the desktop is a GUI object, it derives from the `AnyWin` class. However, unlike other GUI objects, you do not have to record a declaration for the desktop. Instead, you use the predefined identifier `Desktop` when you want to use a method on the desktop.

For example, to call the `GetActive` method on the desktop, you use a call like the following:

```
wActive = Desktop.GetActive ()
```

Related Topics

- *The inheritance embodied in the class hierarchy*
- *Logical classes*
- *CursorClass, ClipboardClass, and AgentClass*

Logical classes

The `AnyWin`, `Control`, and `MoveableWin` classes are logical (virtual) classes that do not correspond to any actual GUI objects, but instead define methods common to the classes that derive from them. This means that SilkTest never records a declaration that has one of these classes.

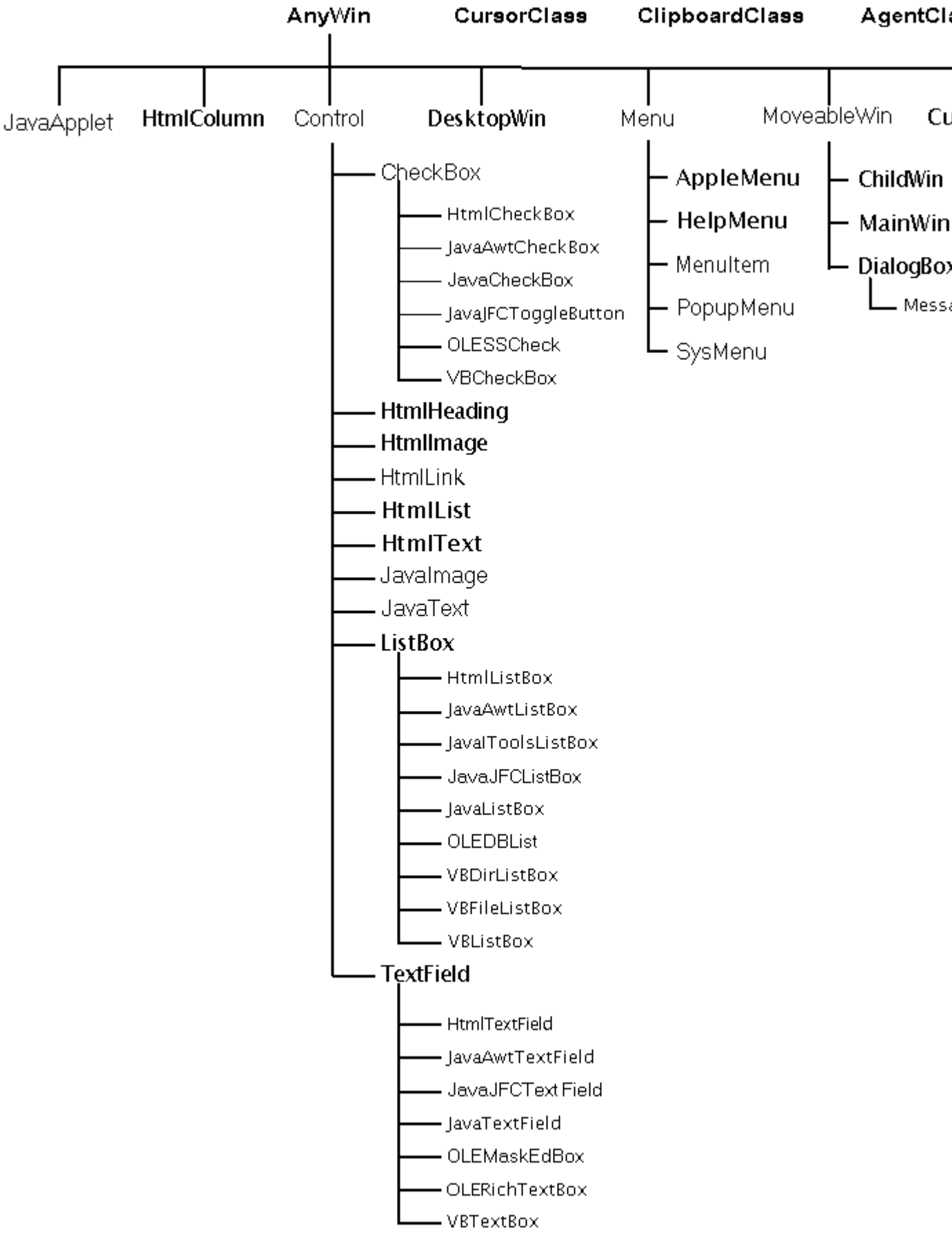
Furthermore, you cannot extend or override logical classes. If you try to extend a logical class, by adding a method, property or data member to it, that method, property, or data member is not inherited by classes derived from the class. You will get a compilation error saying that the method/property/data member is not defined for the window that tries to call it. Nor can you override the class, by rewriting existing methods, properties, or data members. Your modifications are not inherited by classes derived from the class.

Class Hierarchy

You can define your own methods and properties, as well as define your own classes. You can also define your own attributes, which are used in the verification stage in testcases.

The 4Test class hierarchy defines the methods and properties that enable you to query, manipulate, and verify the data or state of any GUI object in your application. You can define your own methods and

properties, as well as define your own classes. You can also define your own attributes, which are used in the verification stage in testcases. The following illustration shows a partial listing of the built-in class hierarchy:


AnyWin
CursorClass
ClipboardClass
AgentCla
JavaApplet
HtmlColumn
Control
DesktopWin
Menu
MoveableWin
Cu
CheckBox
HtmlCheckBox
JavaAwtCheckBox
JavaCheckBox
JavaJFCToggleButton
OLESSCheck
VBCheckBox
HtmlHeading
HtmlImage
HtmlLink
HtmlList
HtmlText
JavaImage
JavaText
ListBox
HtmlListBox
JavaAwtListBox
JavaToolsListBox
JavaJFCListBox
JavaListBox
OLEDBList
VBDirListBox
VBFileListBox
VBListBox
TextField
HtmlTextField
JavaAwtTextField
JavaJFCTextField
JavaTextField
OLEMaskEdBox
OLERichTextBox
VBTextBox
AppleMenu
HelpMenu
MenuItem
PopupMenu
SysMenu
ChildWin
MainWin
DialogBox
Messa

The DOM extension supports four additional classes: `HtmlForm`, `HtmlHidden`, `HtmlMeta`, and `HtmlMarquee`.

Class Hierarchy Inheritance

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The inheritance depicted in the class hierarchy diagram can be summarized as follows:

AnyWin	Any type of GUI object.
BrowserChild	The window that contains the contents of a web page.
Control	Check boxes, combo boxes, list boxes, popup lists, pushbuttons, radio lists, scales, scroll bars, static text (labels), text fields, and so on.
CustomWin	Does not apply. Assigned to objects that do not correspond to any class in the built-in class hierarchy. Because the CustomWin class derives from the AnyWin class, you can use the primitive methods of the AnyWin class, like PressKeys, to define higher level actions to perform on your custom GUI objects.
DesktopWin	The desktop.
HtmlColumn	A column within an HtmlTable in a Web application.
HtmlTable	A series of two or more columns in a Web application.
Menu	Apple menus, help menus, menu items, popup menus, and system menus.
MoveableWin	Child windows (free-standing windows, including MDI sheets) dialog boxes, message boxes, and the application's main window.
TaskbarWin	The taskbar in Windows.

Verifying Attributes and Properties

Attribute Definition and Verification

When you record a testcase, you can verify the testcase using attributes.

You can choose to verify using either attributes or properties. Generally you will verify using properties, because property verification is more flexible. For more information on verifying using properties and attributes, see *Verifying using properties* and *Verifying object attributes*.

For example, the attributes for the `DialogBox` class are `Caption`, `Contents`, `Default button`, `Enabled`, and `Focus`. Here is the 4Test code in the `winclass.inc` file that implements the `Default Button` attribute:

```
attribute "Default button", VerifyDefaultButton, QueryDefaultButton
```

As this 4Test code shows, each attribute definition begins with the statement, followed by three comma-delimited values:

1. The text that you want to appear in the Attribute panel of the Verify Window dialog. This text must be a string.
2. The method SilkTest should use to verify the value of the attribute at runtime.
3. The method SilkTest should use to get the actual value of the attribute at runtime.

Defining a New Attribute for an Existing Class

To add one or more attributes to an existing class, use the following syntax:

```
winclass ExistingClass : ExistingClass...
attribute_definitions
```

Each attribute definition begins with the `attribute` statement, followed by three comma-delimited values:

1. The text that you want to appear in the **Attribute** panel of the **Verify Window** dialog. This text must be a string.
2. The method SilkTest should use to verify the value of the attribute at runtime.
3. The method SilkTest should use to get the actual value of the attribute at runtime.

Each attribute definition must begin and end on its own line. When you define a new attribute, you usually need to define two new methods (steps 2 and 3 above) if none of the built-in methods suffice.

SilkTest allows you to add, delete, or edit the existing functionality of a class; this applies to both functions and variables of a class. However, we recommend that you do not override a function or a variable by declaring a function or variable of that same name. Furthermore, you should never override a variable that has a tag associated with it. You cannot have two variables with the same name in the same level of an object. If you do so, SilkTest will display a compile error.

Defining new verification properties

You can perform verifications in your testcases using properties. These verification properties are different from class properties, which are defined using the property declaration. Verification properties are used only when verifying the state of your application in a testcase. SilkTest comes with built-in verification properties for all classes of GUI objects.

You can define your own verification properties, which will be added to the built-in properties listed in the Verify Window dialog when you record a testcase.

Syntax for attributes

To add one or more attributes to an existing class, use the following syntax:

```
winclass ExistingClass : ExistingClass...
attribute_definitions
```

Each attribute definition must begin and end on its own line.

When you define a new attribute, you usually need to define two new methods if none of the built-in methods suffices.

For example, to add a new attribute to the `DialogBox` class that verifies the number of children in the dialog, you add code like this to your test frame (or other include file):

```
winclass DialogBox:DialogBox

  attribute "Number of children", VerifyNumChild, GetNumChild

  integer GetNumChild()
     return ListCount (GetChildren ()) // return count of children of dialog

  hidecalls VerifyNumChild (integer iExpectedNum)
     Verify (GetNumChild (), iExpectedNum, "Child number test")
```

As this example shows, you use the hidecalls keyword when defining the verification method for the new attribute.

Hidecalls keyword

The keyword `hidecalls` hides the method from the call stack listed in the results. Using `hidecalls` allows you to update the expected value of the verification method from the results. If you do not use `hidecalls` in a verification method, the results file will point to the frame file, where the method is

defined, instead of to the script. We recommend that you use `hidecalls` in all verification methods so that you can update the expected values.

An alternative to NumChildren as a class property

Instead of defining `NumChildren` as a class property, you could also define it as a variable, then initialize the variable in a script. For example, in your include file, you would have:

```
winclass DialogBox : DialogBox
INTEGER NumChild2
 // list of custom verification properties
LIST OF STRING lsPropertyNames = {"NumChild2"}
```

And in your script, before you do the verification, you would initialize the value for the dialog box under test, such as:

```
Find.NumChild2 = ListCount (Find.GetChildren ())
```

Defining Methods and Custom Properties

Defining a new method

To add a method to an existing class, you use the following syntax to begin the method definition:

```
winclass ExistingClass : ExistingClass
```

The syntax `ExistingClass : ExistingClass` means that the declaration that follows extends the existing class definition, instead of replacing it. For more information, see *winclass declaration*.

Defining a new method for a single GUI object

To define a new method to use on a single GUI object, not for an entire class of objects, you add the method definition to the window declaration for the individual object, not to the class. The syntax is exactly the same as when you define a method for a class.

Deriving a new method from an existing one

To derive a new method from an existing method, you can use the derived keyword followed by the scope resolution operator (::).

Use the following syntax:

```
new method : existing method
```

The following example defines a `GetCaption` method for `WPFNewTextBox` that prints the string `Caption as is` before calling the built-in `GetCaption` method (defined in the `AnyWin` class) and printing its return value:

```
winclass WPFNewTextBox : WPFTextBox
GetCaption ()
Print ("Caption as is: ")
Print (derived::GetCaption ())
```

Defining custom verification properties

1. In a class declaration or in the declaration for an individual object, define the variable `lsPropertyNames` as follows: `LIST OF STRING lsPropertyNames`.
2. Specify each of your custom verification properties as elements of the list `lsPropertyNames`. Custom verification properties can be either:
   - Class properties, defined using the property statement.
   - Variables of the class or individual object.

Any properties you define in `lsPropertyNames` will override built-in properties with the same name. With your custom verification properties listed as elements in `lsPropertyNames`, when you record and run a testcase, those additional properties will be available during verification.

Redefining a method

There may be some instances in which you want to redefine an existing method. For example, to redefine the `GetCaption` method of the `Anywin` class, you use this 4Test code:

```
winclass AnyWin : AnyWin
   GetCaption ()
      // insert method definition here
```

Confirming the property list

You can use the `GetPropertyList` method to confirm the list of verification properties for an object. For example, the following simple testcase prints the list of all the verification properties of the Find dialog to the results file:

```
testcase FindDialogPropertyConfirm ()
TextEditor.Search.Find.Pick ()
ListPrint (Find.GetPropertyList ())
Find.Cancel.Click ()
```

Examples

Example Adding a Method to TextField Class

This example adds to the `TextField` class a method that selects all of the text in the text field.

```
winclass TextField : TextField
   SelectAll ()
      STRING sKey1, sKey2
      switch (GetGUIType ())
         case mswnt, msw2003
            sKey1 = "<Ctrl-Home>"
            sKey2 = "<Shift-Ctrl-End>"
         case mswvista
            sKey1 = "<Ctrl-Up>"
            sKey2 = "<Shift-Cmd-Down>"
      // return cursor to 1,1
      this.TypeKeys (sKey1)
      // highlight all text
      this.TypeKeys (sKey2)
```

The keyword `this` refers to the object the method is being called on.

The preceding method first decides which keys to press, based on the GUI. It then presses the key that brings the cursor to the beginning of the field. It next presses the key that highlights (selects) all the text in the field.

For more information on the syntax used in declaring new methods, see *Method declaration*.

Example adding Tab method to DialogBox class

To add a `Tab` method to the `DialogBox` class, you could add the following 4Test code to your `frame.inc` file (or other include file):

```
winclass DialogBox : DialogBox
Tab (INTEGER iTimes optional)
if (iTimes == NULL)
 iTimes = 1
this.TypeKeys ("<tab {iTimes}>")
```

Defining custom verification properties

1. In a class declaration or in the declaration for an individual object, define the variable `lsPropertyNames` as follows: `LIST OF STRING lsPropertyNames`.
2. Specify each of your custom verification properties as elements of the list `lsPropertyNames`. Custom verification properties can be either:
   - Class properties, defined using the property statement.
   - Variables of the class or individual object.

Any properties you define in `lsPropertyNames` will override built-in properties with the same name. With your custom verification properties listed as elements in `lsPropertyNames`, when you record and run a testcase, those additional properties will be available during verification.

Porting Tests to Other GUIs

Handling Differences Among GUIs

Creating a class that maps to several SilkTest classes

Consider the Direction control in the **Find** dialog of the Text Editor application, which allows a user to specify the direction (up or down) of searches. Suppose that this control is implemented as a check box on one GUI, but as a radio list on all other GUIs. As a radio list, the user selects either the Up or the Down radio button. As a check box, the user checks the check box to select Up, and leaves the check box unchecked to select Down.

The first step in solving this portability scenario is to create a new window class that you will use for the object on all platforms. The class you need to define, in effect, generalizes several distinct 4Test classes into one logical class.

To achieve this generalization, you:

- Derive the new class from the 4Test Control class, since both radio lists and check boxes derive from this class.
- Define the class with a GUI-specific tag statement for each platform. Each tag statement states the actual class of the control on the particular GUI. This allows SilkTest to know what the actual class on the control will be at runtime on each of the GUIs.
- Define generalized methods that use a switch statement to branch to the 4Test code that implements the method on the particular GUI.

Here is the class declaration, which is arbitrarily named DirButton:

```
// The class is derived from Control
winclass DirButton : Control
   tag "[RadioList]"
   msw9x tag "[CheckBox]"
   void Select (LISTITEM Item optional)
      BOOLEAN bState
      switch (GetGUIType ())
         case msw9x
            bState = (Item == "Up")
            CheckBox (WndTag).SetState (bState)
         default
            RadioList (WndTag).Select (Item)
```

Note the following:

- The `Select` method acts against the control, regardless of whether it is a RadioList or CheckBox. The method contains a switch statement which executes the `SetState` method if the control is a check box, and the `Select` method if the control is a radio list. The `Select` method also takes an optional parameter, as indicated by the keyword optional.
- Because the tag of the object differ on each GUI, rather than specifying an identifier in the `SetState` and `Select` method calls, you use the `WndTag` property. By doing this, you force SilkTest to construct a dynamic identifier for the object at runtime which will uniquely identify the object as a check box in the one case and as a radio list in all other cases.

The next step is to change your window declarations so that the control has the new class.

Continuing the example, you change the class of the control named `Direction` to **DirButton**.

```
window DialogBox Find
   tag "Find"
   parent TextEditor
   DirButton Direction
      tag "Direction"
```

Creating GUI-specific tags

To close a file on one operating system, you select, for example, **File** > **Quit**, whereas on all other platforms you select **File** > **Exit**. The following window declaration accounts for these differences with two tag statements:

```
MenuItem Exit
   tag "Exit"
   [other OS] tag "Quit"
```

With this declaration, the `Exit` identifier can be used to refer to the menu item regardless of the actual label.

Conditionally loading include files

If you are testing different versions of an application, such as versions that run on different platforms or versions in different languages, you probably have different include files for the different versions. For example, if your applications run under different languages, you might have text strings that appear in windows defined in different include files, one per language. You want SilkTest to load the proper include file for the version of the application you are currently testing.

Load different include files for different versions of the test application

1. Define a compiler constant. For example, you might define a constant named `MyIncludeFile`.
2. Insert the following statement into your 4Test file: `use constant`. For example, if you defined a constant `MyIncludeFile`, insert the following statement: `use MyIncludeFile`. In this example, constant can also be an expression that evaluates to a constant at compile time.
3. When you are ready to compile your 4Test files, specify the file name of the include file you want loaded as the value of the constant in the **Compiler Constants** dialog. Be sure to enclose the value in quotation marks if it is a string.
4. Compile your code.

SilkTest evaluates all compiler constants and substitutes their values for the constants in your code. In this case, the constant `MyIncludeFile` will be evaluated to a file, which will be loaded through the `use` statement.

Deciding which form of tag to use

When an object's caption or label changes on a different GUI, it is usually preferable to use multiple tags, each based on the GUI-specific label or caption, instead of using the index. Not only does it make your declarations easier to understand, but it shields your testcases from changes to the sequence of child

objects. For example, if the Exit item changes so that it is the fourth item and not the fifth, your testcases will still run.

Error messages are different

The `VerifyErrorBox` function, shown below, illustrates how to solve the problem of different error messages on each GUI platform. For example, if a GUI platform always adds the prefix "Error:" to its message, while the other platforms do not, you might use or create a GUI Specifier for that platform and then use the `VerifyErrorBox` function as follows:

```
VerifyErrorBox (STRING sMsg)
   // verifies that the error box has the correct error
   // message, then dismisses the error box

   const ERROR_PREFIX = "ERROR: "
   const ERROR_PREFIX_LEN = Len (ERROR_PREFIX)
   STRING sActMsg = MessageBox.Message.GetText ()

   // strip prefix "ERROR: " from GUI Specifier for that platform error
messages
   if (GetGUIType () == GUI Specifier for that platform)
      sActMsg = SubStr (sActMsg, ERROR_PREFIX_LEN + 1)

   Verify (sActMsg, sMsg)
   MessageBox.Accept ()
```

One logical control can have two implementations

Consider the case where the same logical control in your application is implemented using different classes on different GUIs:

If the kinds of actions you can perform against the object classes are similar, and if SilkTest uses the same method names for the actions, then you do not have a portability problem to address.

For example, the methods for the `RadioList` and `PopupList` classes have identical names, because the actions being performed by the methods are similar. Therefore, if a control in your application is a popup list on one GUI and a radio list on another, your scripts are already portable.

If the two object classes do not have similar methods, or if the methods have different names, then you need to follow the steps outlined in the Related Topics section to port your scripts.

Options sets and porting

Options sets save all current options except General Options. Options sets can be very useful when trying to use the same scripts on different operating systems. The primary differences between the two may be compiler constants.

For example, you might use the compiler constant `sCmdLine`. Usually, the command line to invoke an application differs between the PC operating systems. You could create a compiler constant (note that there is a string limit on compiler constants) for use in the sCmdLine constant to differentiate between the platforms' command lines. You might also use a compiler constant for methods that work slightly differently on the two operating systems, such as the `Pick()` methods.

Specifying options sets

In a testplan, you can specify options sets to be used with the testplan or parts of it. You use options sets to automatically run different tests that require different options without having to manually open options sets.

To ensure that everyone working on a project has the same options settings (such as class mapping), do one of the following:

- Open an Options Set.

- Set these option values at runtime.
- Specify the following statement in the testplan: `optionset: filename.opt`.

Dependent testcases will run with the specified options set opened. The options set will be closed when it passes out of scope. If you don't specify a full path name, the file is considered to be in a directory relative to the directory containing the current testplan or subplan.

Remember:

- Options can also be set at runtime in a test script by using the `Agent` method, `SetOption`, and passing in the name of the option and its value.
- Many Agent options and their values are found in the **Agent Options** dialog.
- Agent options can be set in a `testcase/` function.
- Class map settings, set at runtime, are best set before any tests are executed (for example, in ScriptEnter) and after each testcase (for example TestcaseExit) in case any have been changed in the course of a testcase.
- Class mappings set at runtime using the Agent method `SetOption` are only in effect during test execution; these settings are not available to the recorders.

Supporting differences in application behavior

Although you can account for differences in the appearance of your application in the window declarations, if the application's behavior is fundamentally different when ported, you need to modify your testcases themselves. To modify your testcases, you write sections of 4Test code that are platform-specific, and then branch to the correct section of code using the return value from the GetGUIType built-in function.

This topic shows how to use the GetGUIType function in conjunction with *If statements* and the *Switch statements*.

Switch statements

You can use GUI specifiers before an entire switch statement and before individual statements within a case clause, but you cannot use GUI specifiers before entire `case` clauses.

```
testcase GUISwitchExample()
INTEGER i
FOR i=1 to 5
mswxp, mswnt switch(i)

// legal:
mswxp, mswnt switch (i)
   case 1
      mswxp Print ("hello")
      mswnt Print ("goodbye")
   case 2
      mswxp raise 1, "error"
      mswnt Print ("continue")
   default
      mswxp Print ("ok")

// NOT legal:
switch (i)
   mswxp case 1
      Print ("hello")
   mswnt case 1
      Print ("goodbye")
```

If statements

You can use GUI specifiers in if statements, as long as GUI specifiers used within the statement are subsets of any GUI specifiers that enclose the entire `if` statement.

```
// legal because no GUI specifier
// enclosing entire if statement:
```

```
if (i==j)
   msw32, mswnt Print ("hi")
   msw2000 Print ("bye")

// legal because msw is a subset of enclosing specifier:
msw32, msw2000 if (i==j)
   mswnt Print("hi")

// legal for the same reason as preceding example:
msw32, msw2000 if (i==j)
   Print ("hi")
mswnt else
   Print ("Not the same")

// NOT legal because msw2000 is not a subset
// of the enclosing GUI specifier msw:
msw32 if (i==j)
   msw2000 Print ("bye")  // Invalid GUI type
```

If you are trying to test multiple conditions, then you should use a `select` or `switch` block. You could use nested `if.else` statements, but if you have more than two or three conditions, the levels of indentation will become cumbersome.

You should not use an `if..else if..else` block. Although `if..else if..else` will work, it will be difficult to troubleshoot exceptions that occur because the results file will always point to the first `if` statement even if it was actually a subsequent `if` statement that raised the exception.

For example, in the following testcase, the third string, `Not a date`, will raise the exception:

*** Error: Incompatible types -- 'Not a date' is not a valid date

The exception actually occurs in the lines containing:

```
GetDateTimePart ([DATETIME]sVal, DTP_YEAR) == 2006
```

For the nested `if..else` and the `select` blocks, the results file points to those lines as the sources of the exceptions. However, for the `if..else if..else` block, the results file points to the first `if` statement, in other words to the line:

```
[-] if IsNull (sVal)
```

even though that line clearly is not the source of the exception because it does not concern DATETIME values.

```
[+] testcase IfElseIfElse ()
[-] LIST OF STRING lsVals = {...}
[ ] "2006-05-20"
[ ] "2006-11-07"
[ ] "Not a date"
[ ] STRING sVal
[ ]
[-] for each sVal in lsVals
[-] do
[-] if IsNull (sVal)
[ ] Print ("No date given")
[-] else if sVal == FormatDateTime (GetDateTime (), "yyyy-mm-dd")
[ ] Print ("The date is today")
[-] else if GetDateTimePart ([DATETIME]sVal, DTP_YEAR) == 2006
[ ] Print ("The year is this year")
[-] else
[ ] Print ("Some other year")
[-] except
[ ] ExceptLog ()
[ ]
[-] do
[-] if IsNull (sVal)
```

```
[ ] Print ("No date given")
[-] else
[-] if sVal == FormatDateTime (GetDateTime (), "yyyy-mm-dd")
[ ] Print ("The date is today")
[-] else
[-] if GetDateTimePart ([DATETIME]sVal, DTP_YEAR) == 2006
[ ] Print ("The year is this year")
[-] else
[ ] Print ("Some other year")
[-] except
[ ] ExceptLog ()
[ ]
[-] do
[-] select
[-] case IsNull (sVal)
[ ] Print ("No date given")
[-] case sVal == FormatDateTime (GetDateTime (), "yyyy-mm-dd")
[ ] Print ("The date is today")
[-] case GetDateTimePart ([DATETIME]sVal, DTP_YEAR) == 2006
[ ] Print ("The year is this year")
[-] default
[ ] Print ("Some other year")
[-] except
[ ] ExceptLog ()
[ ]
```

Text field requires Return keystroke

On some GUIs, the `Enter`/`Return` key must be pressed after data is entered into a text field. Suppose you want to create a testcase that enters invalid data into the field, and then checks if the application detects the error. After the testcase enters the invalid data, it needs to use the `GetGUIType` function to determine the GUI, and then press the Return key if the GUI requires it.

For example:

```
// code to enter an invalid string into field
if (GetGUIType () == mswnt)
   MyTextField.TypeKeys ("<Return>")
// code to verify that application detected error
```

Using cross-platform methods in your scripts

In scripts, you can use your cross-platform method names. The window declarations map the cross-platform method names you use in your scripts to the actual methods required to carry out the actions you want on each of the GUIs.

Continuing the example from *Creating a class that maps to several SilkTest classes*, you use the `Select` method in your code to select the control named `Direction`.

```
testcase SearchBackward ()

   LISTITEM Item
   Item = "Up"
   Find.Invoke ()
   Find.Direction.Select (Item)
   .
   .
   .
   Find.Dismiss ()
```

Note that the script does not indicate that anything unusual is happening. All of the steps necessary to make the Select method work properly, regardless of the class of the object, are encapsulated in the class and window declarations.

Using the index as the tag

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you are certain that an object's position in relation to its sibling objects of the same class will remain the same when the application is ported, you can use the index form for the tag.

Repeating the example from the preceding section, because the Exit/Quit menu item is the fifth menu item on the **File** menu (on all platforms), you can use the index form for the tag (#5) as shown here:

```
MenuItem Exit
   tag "#5"
```

About GUI Specifiers

Class declarations

Be careful using GUI specifiers before class declarations; they can be ambiguous. Any ambiguities must be resolvable at compile-time.

```
// bad style:
msw winclass myclass
mswnt winclass myclass
window myclass inst // Ambiguous. Is it an instance of
                    // the msw class or the mswnt class?
```

The preceding example's ambiguity can be resolved by specifying a GUI target with conditional compilation (so that, for example, only code for msw gets compiled, in which case inst would be an instance of the msw class or by explicitly using a GUI specifier for the window, as follows:

```
// good style:
msw winclass myclass
mswnt winclass myclass
msw window myclass inst
```

Conditional compilation

If you have GUI-specific code in your scripts and declarations, you can have SilkTest conditionally compile your code based on the values of the GUI specifiers - only code specific to a particular GUI is compiled (as well, of course, as all code that is not GUI-specific). This has two advantages:

- The compilation is faster
- The resulting code is smaller and require less memory to run

You can also cause conditional compilation by using constants, which are evaluated at compile time.

Constants are not restricted to conditional compilation. You can use constants for any value that you want resolved at compile time.

Conditionally compile code

1. Prefix any 4Test statements that are GUI-specific with the appropriate GUI specifier.
2. Specify the platforms that you want to compile for by entering the appropriate GUI specifiers in the GUI Targets field in the Runtime Options dialog. You can specify as many GUI targets as you want; separate each GUI specifier by a comma.

   Setting a GUI target affects which classes are listed in the **Library Browser**.
3. To conditionalize code based on the value of constants you define, do the following:
   1. Click the **Compiler Constants** button in the **Runtime Options** dialog.
   2. The **Compiler Constants** dialog is displayed.
   3. Define a constant and specify its value.

4. Use the constant in your code anywhere you can specify an expression.

4. Click **OK** to close the **Runtime Options** dialog.

GUI with inheritance

When using GUI specifiers for parent classes, you must explicitly use the GUI specifiers with the descendants:

```
mswxp winclass newclass
mswxp winclass subclass : newclass
mswxp window subclass inst
```

GUI with global variables

Be careful when using GUI specifiers with global variables: SilkTest initializes global variables before connecting to an Agent. This might not give you the results you want if you are doing distributed testing.

Let's say that you are running tests on a remote machine that is listed in the Runtime Options dialog. Because SilkTest initializes all global variables before connecting to an Agent, any GUI specifier at the global level will initialize to the host machine, not the target machine you want to test against.

For example, say the host machine is running Window 2003 and the target machine is running Windows 2000. Consider the following script:

```
mswxp STRING sVar1 = SYS_GetEnv("UserName")

mswxp STRING sVar1 = SYS_GetRegistryValue
   (HKEY_LOCAL_MACHINE,     "System\CurrentControlSet\Control", "Current
User")

main()
   print(sVar1)
```

This script fails, with the error message:

```
*** Error: Registry entry 'Current User' not found
```

because `sVar1` is initialized to the value for the host system (the GUI specifier msw98), not the target system (msw2000).

Constants behave similarly to global variables if you use a GUI specifier to initialize the variable (or constant). It is a good idea to use GUI specifiers in the main function, under Options/Runtime or another function that is called after the Agent is contacted.

Marking 4Test Code as GUI Specific

Using SilkTest, you can create portable testcases that will test your application on any of the supported GUIs. The reason for this is that your testcases use logical names, called `identifiers`, to refer to the GUI objects, and not actual names, called tags. Therefore, if there are differences in the ported application's appearance, you need only change the window declarations, not the testcases themselves.

The porting scenarios described section use 4Test keywords called GUI specifiers to indicate that portions of include files or script files are specific to a particular GUI. Before studying these scenarios, you should understand which GUI specifiers are available and how to use them in your include files and script files.

4Test includes a long list of GUI specifiers.

Syntax of a GUI specifier

A GUI specifier has this syntax:

```
[[gui-type [,gui-type]] | [!gui-type]]
```

`gui-type` is the GUI. You can express this in one of two mutually exclusive ways. For example, you can specify one or more GUIs separated by commas, as in:

```
mswxp, mswin7
```

Or you can specify all but one GUI, as in the following, which indicates that what follows applies to all environments except Windows NT-based operating systems:

```
! mswnt
```

What happens when the code is compiled

Only code relevant to the GUI environments specified in the GUI Targets field (plus all common code) will be compiled. If you do not list any GUI specifiers in the GUI Targets field, all code will be compiled; at runtime, code not relevant to the platform the application is running on will be skipped.

The constants you have defined are evaluated and used to compile the code. You can use this feature to conditionally load include files.

Where you use GUI specifiers

A GUI specifier can appear before any 4Test declaration or statement except the `use` statement, which must be evaluated at compile time, with the following exceptions.

- `Switch` statements
- `If` statements
- `Type` statements
- `Do… except` statements
- `Class` declarations
- GUI with inheritance
- GUI with global variables

If you try to use a browser specifier instead of a GUI specifier to specify a window, SilkTest will generate an error. The primary use of browser specifiers is to address differences in window declarations between different browsers. Each Agent connection maintains its own browser type, allowing different threads to interact with different browsers.

Do...except statements

You can use GUI specifiers to enclose an entire `do...except` statement and before individual statements, but you cannot use GUI specifiers before the `except` clause.

```
// legal:
do
   mswxp Verify (expr1,expr2)
   mswin7 Verify (expr3,expr4)
except
   mswin7 reraise
   mswxp if (ExceptNum () == 1)
       Print ("err, etc.")
// NOT legal:
mswin7 do
   Verify (expr,expr)
mswxp except
   reraise
```

Type statements

You can use a GUI specifier before a type `type ... is` enum or `type ... is` set statement, but not before an individual value within the type declaration.

Supporting GUI-specific Objects

Supporting GUI-specific captions

SilkTest, by default, bases the tag for an object on the object's actual caption or label. If the captions or labels change when the application is ported to a different GUI, you have two options:

- You can have multiple tags, each based on the platform-specific caption or label.
- You can have a single tag, using the index form of the tag, as long the relative position of the object is the same in the ported versions of the application.

Then, in your testcases, you can use the same identifier to refer to the object regardless of what the object's actual label or caption is.

Supporting GUI-specific executables

The command to start the application will almost always be different on each GUI. SilkTest's Invoke method expects to find the command in the constant sCmdLine, which is defined in your application's main window declaration. You should declare as many sCmdLine variables as there are GUIs on which your application runs, beginning each declaration with the appropriate GUI specifier.

For example, the following constants specify how SilkTest should start the Text Editor application on Windows and Windows Vista:

```
msw32 const sCmdLine = "c:\program files\<SilkTest install directory>\silktest
\textedit.exe"
mswvista const sCmdLine = "{SYS_GetEnv('SEGUE_APPS')}/SilkTest/demo/textedit"
```

Supporting GUI-specific menu hierarchies

When an application is ported, there are two common structural differences in the menu hierarchy:

- The menu bar contains a platform-specific menu.
- A menu contains different menu items.

To illustrate the case of the platform-specific menu, consider the Microsoft Windows system menu or a Vista menu (for example). SilkTest recognizes these kinds of standard GUI- specific menus and includes the appropriate GUI specifier for them when you record declarations.

For menus that SilkTest does not recognize as platform-specific, you should preface the window declaration with the appropriate GUI specifier.

Different menu items - example

To illustrate the case of different menu items, suppose that the Edit menu for the Text Editor application has a menu item named Clear which appears on the Windows version only. The declaration for the Edit menu should look like this:

```
Menu Edit
   tag "Edit"
   msw32 MenuItem Clear
      tag "Clear"
   MenuItem Undo
      tag "Undo"
```

Supporting Custom Objects

Why SilkTest sees objects as custom windows

An object is defined by:

- Its actual class name
- The underlying software code that creates and manipulates it

SilkTest defines an object (window or control) as a `CustomWin` whenever either or both of these definitions vary from what is considered to be standard. While you record window declarations, SilkTest attempts to identify the class of each object in your GUI and assign the appropriate class from the built-in class hierarchy. If an object does not correspond to one of the built-in classes, SilkTest designates the object as a custom window and assigns it to the class `CustomWin`.

For example, SilkTest supports the standard MFC library, which is a library of functions that allow for the creation of objects and the mechanism of interaction with them. In supporting these libraries, SilkTest contains algorithms to interrogate the objects based upon the standard libraries. When these algorithms do not work, SilkTest reports the object as a `CustomWin`.

Let's say that you see a text box in a window in your application under test. It looks like a normal text field, but SilkTest calls it an object of the class `CustomWin`.

Two reasons why SilkTest sees the object as CustomWin

There are two reasons why SilkTest sees the object as a `CustomWin`.

- Upon its definition in the application under test, the control was simply named differently than the standard name. For example, instead of it being named a **TextField**, it was named **EnterTextRegion**. If this is the only reason, then you can class map the control to the standard name.
- You never know whether class mapping will work until you try it. It will work if the object isn't really a custom object, but rather a standard control with a non-standard name. Try this as your first attempt at dealing with a `CustomWin`.

If the class mapping does not work, it is because of the following reason.

- The object truly is a custom object; that is, the software in the application under test that creates and manipulates the object is not from the standard library. That means that the SilkTest algorithms written to interrogate this kind of object will not work, and other approaches will have to be used to manipulate the object.

How you support custom objects depends on whether the object is a graphical control, such as a tool bar, or whether it is not a graphical control, such as a text box.

Working with Custom Objects

Mapping custom classes to standard classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When an object shows up in the **Record Window Declarations** dialog as a `CustomWin`, but is actually a standard GUI object which the application developers have renamed, you should map the custom class name to the built-in class name while in the dialog.

Among the standard classes the SilkTest displays in the Class Map dialog, there are three classes that can be described as "meta" classes:

- `BUTTON` causes the Agent to treat the object as a kind of button, whether it be an instance of `PushButton`, `CheckBox`, or `RadioButton`. The kind of button depends on the object's style bits.

- `STATIC` causes the Agent to treat the object as Static Text if the appropriate style bits (for example, `SS_LEFT` and `SS_CENTER`) are set. Otherwise, only the methods of the `AnyWin` class apply.
- `MDI` client windows are containers that sit between application frame windows and MDI document windows. Mapping a custom object to `MDICLIENT` means you do not need a tag for it in order to refer to one of its children. You cannot perform operations on them.

Map a custom class

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When a declaration for a CustomWin appears in the Record Window Declarations dialog:

1. Press `Ctrl+Alt`. The declarations are frozen in the Window Declaration list box in the lower half of the **Record Window Declarations** dialog.
2. In the **Window Declarations** list box, click the line containing the declaration for the custom object. The line is highlighted and the declaration for the `CustomWin` appears in the **Window Detail** group box.
3. In the **Window Detail** group box, click **Class Map**. The **Class Map** dialog appears. The name of the custom class is displayed in the **Custom Class** text field.
4. In the **Standard Class** field, enter the name of the built-in 4Test class to which you are mapping the custom class.
5. Click **Add**. SilkTest adds the class name.
6. Click **OK**.

When you resume recording, the object has the standard 4Test class. If SilkTest encounters a similar object, it automatically maps the object to the correct 4Test classes. You must modify any prerecorded declarations containing these objects to use the standard class.

Perform a class mapping when a declaration for a CustomWin appears in the Record Window declaration dialog

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Press `Ctrl+Alt`. The declarations are frozen in the **Window Declaration** list box in the lower half of the **Record Window Declarations** dialog.
2. In the Window Declarations list box, click the line containing the declaration for the custom object. The line is highlighted and the declaration for the CustomWin appears in the Window Detail group box.
3. In the **Window Detail** group box, click **Class Map**. The **Class Map** dialog appears. The name of the custom class is displayed in the **Custom Class** text field.
4. In the **Standard Class** field, enter the name of the built-in 4Test class to which you are mapping the custom class.
5. Click **Add** and then click **OK**.

When you resume recording, the object has the standard 4Test class. Any subsequent similar objects encountered will automatically be mapped to the correct 4Test class. You must modify any prerecorded declarations containing these objects to use the standard class.

Options for nongraphical custom controls

If your application uses a non-graphical control that does not map to any of those supported by SilkTest, you have these options:

- If you have the SilkTest Extension Kit, you can add full support for the custom object to SilkTest. Refer to the extension kit documentation for more information.
- If the application's developer created DLLs to interact with the custom object, you can call the DLL functions from a script.
- Otherwise, you can add partial support for the custom object by creating a new class derived from `AnyWin` and writing methods that use the primitive methods of the `AnyWin` class, like `TypeKeys` and

`MoveMouse`, to implement, as much as is possible, the methods appropriate for the non-graphical control.

Adding xy coordinates to a declaration

You can record the x, y locations of a graphical control, such as a toolbar, and include it in a window declaration.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. Position the cursor in the window declaration at the end of the tag to which you want to add coordinates.
2. Type a slash character `/`.
3. Click **Record** > **Window Locations** to open the **Record Window Locations** dialog.
4. Track the cursor over the object. The dialog displays the name of the object and its x,y coordinates relative to the screen, the frame (the main window and its window decoration), and the client (the main window minus its window decoration).
5. Press `Ctrl+Alt` to freeze the declaration.
6. Since this procedure is appending the location to a window declaration, click the **Client option** button.
7. Click **Paste to Editor**, and then click **Close**.

Modify declarations for each of the icons contained in an evenly sized and spaced tool bar

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

1. In the window declarations file, make as many copies of this recorded declaration as there are discrete objects.
2. You can retain the original class (`CustomWin`) if the functionality inherited from the `AnyWin` class is sufficient. Or you can specify the name of a class you create that contains the added functionality you need.
3. Change the identifier to some string that represents the icon's action.
4. Append the tag with the icon's location suffix in the tool bar. You express the location using this syntax:
   ```
   (column:total-columns, row:total-rows)
   ```
   For example, you specify the icon in the third column, first row, like this:
   ```
   (3:26, 1:1)
   ```
   You append this location to the tag with the forward slash (/) character.

Adding a location suffix to the declarations tag

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can, however, create declarations for each discrete object. To do this, make as many copies of the original recorded declaration as there are discrete objects. Then add a location suffix to the tag in each declaration, which is the location of the object within the graphical area.

SilkTest provides two ways to specify the location suffix of contained graphical objects, depending on the size and spacing of the control:

- Controls that are sized and spaced evenly in a grid.
- Controls that are irregularly sized and spaced.

Each of these are described below.

Evenly sized and spaced controls

If a group of graphical controls are equal in size and evenly spaced in a grid, you can specify the location of each control as column y of the total number of columns and row x of the total number of rows. This syntax is both cross-platform and resolution independent.

Irregularly sized or spaced controls

If the graphical controls in a group are not the same size or are not evenly spaced in a grid, you need to specify in the declaration the location suffix of each control as an exact x,y point. This x,y point typically corresponds to the center of the object. This syntax is not necessarily cross-platform or resolution independent.

You specify a location in its declaration as an x,y coordinate using the following syntax:

(x, y)

You append this location to the tag with the forward slash (/) character.

SilkTest Does Not Recognize the Class of an Object

While using the **Record Window Declarations** dialog, you will on occasion notice that the recorded class is `CustomWin`, indicating that SilkTest does not recognize the class of the object. The object is interpreted as a custom object. Custom objects often look and act the same as their corresponding known, standard objects and they may correspond to built-in, known classes

If the object is in fact a custom object, to be able to record and run testcases that interact with the custom object, see *Mapping custom classes to standard classes*.

However, if the object is not actually a custom object, but is instead a standard object that your application's developers have renamed, you can record and run testcases merely by establishing a class map between the renamed class and the standard 4Test class. You can also filter out unneeded custom classes can be filtered from the class hierarchy.

Class mapping only works for objects that are created with standard API calls but are given non-standard names.

Supporting custom text fields

Suppose your application has an object that acts like a text field, but which is not implemented using your GUI's standard text field object. The following example illustrates how you can derive a new class from `AnyWin` and define methods for the custom object. The example defines the `ClearText`, `GetMultiText`, `SetMultiText`, and `GetMultiSelText` methods.

```
// This new class defines methods that re-implement
// the methods of the TextField class so that they will
// work on custom text fields. To be able to use these
// methods, you must change the class of object in the
// declarations from CustomWin to CustomTextField.

winclass CustomTextField : AnyWin

   // This method clears the text field by moving the
   // cursor to the start of field, selecting the text
   // to the end of the file, and deleting the selected
   // text

   void ClearText ()
      TypeKeys ("<Ctrl-Home>")
      TypeKeys ("<Ctrl-Shift-End>")
      TypeKeys ("<Backspace>")

   // This method writes text to the text field.
   // It first calls ClearText and then uses TypeKeys to
```

```
   // input the text passed in.

   void SetMultiText (STRING sText)
      ClearText ()
      TypeKeys (sText)

   // This copies the currently selected text to the
   // clipboard and returns the clipboard contents.

   LIST OF STRING GetMultiSelText ()
      Clipboard.SetText () // Clear the clipboard
      TypeKeys ("<Ctrl-Insert>")
      return (Clipboard.GetText ())

   // This method highlights all of the text in the
   // text field, copies the highlighted text to the
   // clipboard, and returns the clipboard contents.

   LIST OF STRING GetMultiText ()
      Clipboard.SetText () // Clear the clipboard
      TypeKeys ("<Ctrl-Home>")
      TypeKeys ("<Ctrl-Shift-End>")
      TypeKeys ("<Ctrl-Insert>")
      TypeKeys ("<Left>")
      return (Clipboard.GetText ())
```

Supporting custom list boxes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

To support custom list boxes, you can write "low-level" methods. Or, a second approach is to implement the necessary Microsoft Windows messages and set the necessary Windows style bits so that you can use the standard list box methods on your custom list box. The Microsoft Windows messages which you must implement are:

- `LB_GETCOUNT`
- `LB_GETTEXT`
- `LB_GETITEMRECT`
- `LB_GETTOPINDEX`
- `LB_GETSEL`
- `LB_GETTEXTLEN`

And the Windows style bits that you must set are:

- `WS_VSCROLL`
- `LBS_EXTENDEDSEL`
- `LBS_MULTIPLESEL`

Supporting graphical controls

If an application contains a graphical area, for example a tool bar, which is actually composed of a discrete number of graphical objects, SilkTest records a single declaration for the entire graphical area; it does not understand that the area contains individual objects.

Using Clipboard Methods

Get and Set Text Sample Code

The following sample code retrieves the contents of the `CustomTextBuffer` by placing it on the **Clipboard**, then printing the **Clipboard** contents:

```
// Go to beginning of text field
CustomTextBuffer.TypeKeys ("<Ctrl-Home>")
// Highlight it
CustomTextBuffer.TypeKeys ("<Ctrl-Shift-End>")
// Copy it to the Clipboard
CustomTextBuffer.TypeKeys ("<Ctrl-Insert>")
// Print the contents of the Clipboard
Print (Clipboard.GetText())
```

Setting text

Similarly, the following sample code inserts text into the custom object by pasting it from the Clipboard:

```
// Go to beginning of text field
CustomTextBuffer.TypeKeys ("<Ctrl-Home>")
// Highlight it
CustomTextBuffer.TypeKeys ("<Ctrl-Shift-End>")
// Paste the Clipboard contents into the text field
CustomTextBuffer.TypeKeys ("<Shift-Insert>")
```

You can wrap this functionality in `GetText` and `SetText` methods you define for your custom class, similar to what was shown in supporting custom text fields.

Modified declaration use

In a script, to use the `Click` method on this icon, you use this command:

```
app-name.FileOpen.Click ()
```

Using the modified declaration

Once you create window declarations like these for the graphical objects in your application, you can manipulate them as you would any other object. For example, if the tool bar was contained in an application named `MyApp`, to click on the **FileOpen** icon in the tool bar, you use this command:

```
MyApp.FileOpen.Click ()
```

You need to write this statement, and others that access the objects declared above, such as Save and Printer, by hand. **Record** > **Testcase** and **Record** > **Actions** will not use these identifiers.

Using Clipboard methods

If you are having trouble getting or setting information with a custom object that contains text, you might want to try the 4Test Clipboard methods. For example, assume you have a class, `CustomTextBuffer`, which is similar to a `TextField`, but using the `TextField`'s `GetText` and `SetText` methods do not work with the `CustomTextBuffer`. You can use the `ClipboardClass` `GetText` and `SetText` methods.

Filtering Custom Classes

Using Class Mapping to Filter Custom Classes

You can use 4Test to filter out unnecessary classes, such as invisible containers. Ignoring these unnecessary classes simplifies the object hierarchy and shortens the length of the lines of code in your test scripts and functions. Container classes or 'frames' are common in GUI development, but might not be necessary for testing.

You enable SilkTest to ignore instances of a container class through the **Class Map** dialog box. When you enable SilkTest to ignore these classes, the classes are not recorded.

You can suppress the controls for certain classes for .NET, Java SWT, and Windows API-based applications. Other extensions do not support this type of class-mapping for window classes, so you must modify the extension .ini file in order to ignore window classes in the Java or ActiveX/VB extensions.

After filtering out a custom container class, you may lose the ability to see its child objects.

You can also map a class to LookUnder to look 'through' the class, seeing the objects under it. For example, there is a `BlackFrame` class in Visual Basic that is a 3-D black border around controls. LookUnder is not in the standard classes list, so if you want to use it you must type it in.

You can use class mapping to filter out the following:

- Object layers to ignore non-logical windows.
- Extra or "invisible" window layers in the object hierarchy.
- An overlaying window that obstructs an object under it.

Overview of Style-Bits

This functionality is available only for projects or scripts that use the Classic Agent.

Classes may allow different styles of instantiation. Style-bits determine the styles that can be applied to an object. For example, a PushButton, CheckBox, and RadioButton are all variants of the native Windows Button class. They are all the same class, but each has different style-bit to determine the specific look and behavior of the button.

You can map not only an entire class, but also specific 'styles' of one class to another known class.

Class Mapping with Style-Bits

This functionality is available only for projects or scripts that use the Classic Agent.

1. In the **Custom Class** text box of the **Class Map** dialog box, enter the class name, style-bit, and style-mask.
   The style-mask tells SilkTest which parts of the style-bit to use. As a general rule, repeat the style-bit.
2. In the **Standard Class** text box, enter the known class.
   This can be any standard 4Test class or a user-defined class.
3. Click **Add** and then click **OK**.

Note: The style-mask can also take the format `0xFFFFFFFF`, where 'F' includes the bit and '0' turns it off. For example, one custom PageList might look like `0x12345678` and another of the same type might look like `0x12345679`. You can class map it like:

```
newpageclass,0x12345678,0x12345678=PageList
newpageclass,0x12345679,0x12345679=PageList
```

And on and on for each one like it, or…

```
newpageclass,0x12345678,0xFFFFFFF0=PageList
```

What this says is that every instance of `newpageclass 0x1234567?` should be class mapped to PageList. The last bit, being turned off, is ignored.

OCR Support

OCR Support

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest provides a 4Test include file, `OCR.inc`, which contains two 4Test functions that are used to perform optical character recognition (OCR). One function converts bitmap files to text. The other allows

you to pass in a window identifier and extract the text from the window (or a region of the window). To use the 4Test OCR functions, include `OCR.inc` in your test script or include file, or add the include file through the **Use Files** text box in the **Runtime Options** dialog box. To include the function documentation in the library browser, add `OCR.txt` through the **Help files for library browser** text box in the **General Options** dialog box.

The 4Test functions call functions in a Silk DLL file that extends the third-party Textract DLL file from Structu Rise. The Textract DLL uses a font pattern database file to recognize text of certain sizes and styles for fonts that are specified in an initialization file. Although a default version of the font pattern database is installed with SilkTest, we recommend that you configure the font pattern database to include the fonts used by your application. For additional information, see *Generating the Font Pattern Database*.

The SilkTest OCR Module

This functionality is available only for projects or scripts that use the Classic Agent.

The following files comprise the OCR module provided with SilkTest. All of the files must reside in the SilkTest directory:

Exgui.exe	Utility for generating the font pattern database. Can also be used as a standalone utility for text recognition.
OCR.inc	The 4Test include file that provides high-level 4Test functions based on the functions in `SgOcrLib.dll`.
SgOcrLib.dll	Borland extension of `Textract.dll` that provides high-level functions.
SgOcrLib.inc	Declares the functions in `SgOcrLib.dll` so that they can be called in 4Test.
SgOcrPattern.pat	Font pattern database that controls text conversion.
Textract.dll	Textract OCR DLL provided by Structu Rise.
Textract.ini	Initialization file for the Textract OCR DLL. The following two sections contain settings that may require modification: • In the `[Options]` section, the `Database Path` setting must point to the OCR pattern file, `<SilkTest directory>\SgOcrPattern.pat`. • The `[Recognition]` section contains settings that control which fonts are used to generate the pattern file.

The 4Test OCR Functions

This functionality is available only for projects or scripts that use the Classic Agent.

`OCR.inc` includes the following 4Test functions for OCR:

Function	Description	Parameters		Syntax
`OcrGetTextFromBmp`	Converts a bitmap file into text. The conversion uses the pre-configured pattern file, which is specified in the `Database Path` setting in the `textract.ini` file.	**iRet**	Result length. If conversion fails, then an E_OCR exception will be raised. INTEGER.	`iRet` = `OcrGetTextFromBmp` (`sOcrText`, `sBitmapFile`)
		sOcrText	The result of converting	

Function	Description	Parameters		Syntax
			the bitmap to text. NULL if conversion failed. OUT. STRING.	
		sBitmapFile	The bitmap (.bmp) file to convert. STRING.	
`OcrGetTextFromWnd`	Converts a bitmap of a window, or an area within a window, into text. The conversion uses the pre-configured pattern file, which is specified in the `Database Path` setting in the `textract.ini` file.	**iRet**	Result length. If conversion fails, then an E_OCR exception will be raised. INTEGER.	`iRet = OcrGetTextFromWnd (sText, wWindow[, rCapture])`
		sText	The result of converting a bitmap of the window to text. NULL if conversion failed. OUT. STRING.	
		wWindow	The window that will be the source of the bitmap to be converted to text. WINDOW.	
		rCapture	The capture region. OPTIONAL. RECT.	

Example

The sample test script (ocrtest.t) includes a testcase (shown below) that extracts the text from a Microsoft Word document. Microsoft Office controls are recognized as custom windows (CustomWin) by SilkTest, so you cannot use the 4Test GetText() method to get the text. However, you can use the OcrGetTextFromWnd() function to

capture a bitmap of the document window and convert it to text. Notice that, if necessary, the testcase will scroll through the document and capture multiple bitmaps.

```
testcase GetOcrAPIDocText (STRING sDocument)
  MSWord.SetActive ()

  // Open the specified document.
  MSWord.OpenDoc (sDocument)

  // Capture each page in succession.
  // Start at the top and page down until the bottom is reached
  LIST OF STRING lsResults = {}
  STRING sResult = NULL
  INTEGER iMaxPos, iCurPos, iLastPos = -1
  INTEGER iResLen, iTotalLen = 0

  withoptions
    BindAgentOption (OPT_REQUIRE_ACTIVE, FALSE)
    BindAgentOption (OPT_VERIFY_ACTIVE, FALSE)
    TheDoc.ScrollBarV.ScrollToMin ()
    iCurPos = TheDoc.ScrollBarV.GetPosition ()
    iMaxPos = TheDoc.ScrollBarV.GetRange ().iMax

  while TRUE
    // If we are capturing the first page, then eliminate the
flashing cursor
    // by highlighting the current character. Otherwise, page
down to capture
    // the next page.
    MSWord.SetActive ()
    if sResult == NULL
      // First page
      MSWord.TypeKeys ("<Shift-Right>")
    else
      // Page down
      withoptions
        BindAgentOption (OPT_REQUIRE_ACTIVE, FALSE)
        BindAgentOption (OPT_VERIFY_ACTIVE, FALSE)
        TheDoc.ScrollBarV.ScrollByPage (1)
        iLastPos = iCurPos
        iCurPos = TheDoc.ScrollBarV.GetPosition ()
      // If scrolling did not change the scrollbar position,
then
      // we have reached the bottom. Also, if we scrolled up
instead
      // of down by paging down, then we have reached the
bottom.
      if iCurPos <= iLastPos
        break

    // Convert the bitmap for the current view.
    iResLen = OcrGetTextFromWnd (sResult, TheDoc.CurrentView)
    if sResult != NULL
      ListAppend (lsResults, sResult)
      iTotalLen = iTotalLen + Len (sResult)

  ResPrintList ("Document text ({iTotalLen} chars)", lsResults)
```

If neither of the 4Test functions in `OCR.inc` provides the functionality that you need, you can call the DLL functions in the Borland DLL, `SgOcrLib.dll`, through directly using the function declarations in `SgOcrLib.inc`. The DLL functions are documented

at the top of SgOcrLib.inc. The functions in OCR.inc can serve as an example of how to use the DLL functions.

Instructions for Generating the Font Pattern Database

This functionality is available only for projects or scripts that use the Classic Agent.

Use the Exgui.exe utility to generate the pattern file SgOcrPattern.pat. The text conversion is performed using Windows fonts. Before conversion, the required fonts must be processed into the pattern file. Before generating the pattern file, the required fonts, font sizes, and font styles must be configured in the Textract.ini file. Open Textract.ini and adjust the following settings in the [Recognition] section:

Include1	List of fonts that are to be converted.
Exclude	List of fonts that are not to be converted.
Italic	1 - Convert italic characters; 0 - Exclude italic characters.
Bold	1 - Convert bold characters; 0 - Exclude bold characters.
Underlined	1 - Convert underlined characters; 0 - Exclude underlined characters.
Sizes	Range of font sizes, <min>-<max>, that are to be converted.

Once Textract.ini has been configured, open the Exgui.exe application and click **Build font pattern database**. When the **Textract - Build Font Base** dialog box opens, click **OK** and wait for the pattern file to be generated. The file name and path will be saved based on the Database Path setting in the [Options] section of the textract.ini file.

More Information about SGOCRLIB.DLL

This functionality is available only for projects or scripts that use the Classic Agent.

Copy the following files into the directory where the executable that calls the dll, for example Partner.exe, resides:

- SgOcrLib.dll
- SgOcrPattern.pat
- Textract.dll
- Textract.ini

For convenient pattern file generation, we recommend that you also copy the exgui.exe file to this directory.

Open the Textract.ini file and modify the Database Path setting in the [Options] section to point to the correct location of SgOcrPattern.pat.

Have the application, for example SilkTest, load SgOcrLib.dll, which contains the functions explained in the following section.

Pattern File Generation

This functionality is available only for projects or scripts that use the Classic Agent.

Use the exgui.exe application to generate the appropriate pattern file. The text conversion is performed using Windows fonts. The required fonts must be processed into the pattern file before any conversion is performed. The required fonts, font sizes, and font styles must first be configured in the textract.ini file. Open textract.ini and adjust the following settings:

Include1	List of fonts that are to be converted.

Exclude	List of fonts that are not to be converted.
Italic	1 - Convert italic characters; 0 - Exclude italic characters.
Bold	1 - Convert bold characters; 0 - Exclude bold characters.
Underlined	1 - Convert underlined characters; 0 - Exclude underlined characters.
Sizes	Range of font sizes, <min>-<max>, that are to be converted.

Then open the `exgui.exe` application and click **Build font pattern database**. Click **OK** and wait for the pattern file to be generated. The file name and path will be saved based on the `Database Path` setting in the `textract.ini` file.

Support Internationalized Objects

Overview of SilkTest support of Unicode content

SilkTest is Unicode-enabled, meaning SilkTest is able to recognize double-byte (wide) languages. We have enabled components within the application to deal with Unicode content. The SilkTest GUI supports the display and input of wide text. The 4Test language processor has been enhanced to support wide text. All 4Test library functions have been widened. The extensions have been enhanced to support the input and output of wide text.

We have added and modified 4Test functions to deal with internationalization issues. With SilkTest you can test applications that contain content in double-byte languages such as Chinese, Korean, or Japanese (Kanji) characters, or any combination of these. You can also name SilkTest files using internationalized characters. SilkTest supports three text file formats: ANSI, Unicode and UTF-8.

SilkTest supports:

- localized versions of Windows*
- international keyboards and native language Input Method Editors (IME)
- passing international strings as parameters to testcases, methods, etc. and comparing strings
- accessing databases, via direct ODBC standard access
- reading and writing text files in multiple formats: ANSI, Unicode, and UTF-8

* For information on the currently supported browsers, see the Release Notes by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

Before testing double-byte characters with SilkTest

Testing an internationalized application, particularly one that contains double-byte characters, is more complicated than testing an application that contains strictly English single-byte characters. Testing an internationalized application requires that you understand a variety of issues, from operating system support, to language packs, to fonts, to working with IMEs and complex languages.

Before you begin testing your application using SilkTest, you must:

- meet your application under test's (AUT) needs for any necessary localized OS, regional settings/ options configuration, and/or required language packs
- install the fonts necessary to display your AUT
- if you are testing an application that requires an IME for data input, install the appropriate IME

Using DB Tester with Unicode content

To use DBTester with Unicode characters:

- You must have a Unicode-capable driver (ODBC version 3.5 or higher) associated with the data source name you are using in your testplan.
- The database must be Unicode capable (SQL Server 7 and 2000, Oracle 8 and higher).

Issues displaying double-byte characters

When you are dealing with internationalized content, being able to display the content of your application is critical. Carefully consider the following:

- The operating system - to display double-byte characters in the system dialogs and menus used by your application, your OS needs to be capable of displaying this content.
- SilkTest - there are two components of SilkTest where you need to be concerned about displaying your content: the SilkTest Editor and the SilkTest dialogs. For information see Displaying Double-byte Characters in SilkTest.
- The application under test - you will need to have a font installed that is capable of displaying the content of your application. If you have multiple-languages represented in your application, you will need a font that spans these languages.
- The browser - if your application is web-based you should make sure that you are using a browser that supports your content, that it is configured to display your content and that you have fonts installed necessary to display your application.
- Complex scripts (languages) - SilkTest does not support complex scripts such as the bi-directional languages Hebrew and Arabic. These are languages that require special processing to display and edit because the characters are not laid out in a simple linear progression from left to right, as are most western European characters. For more information see Working with Bi-directional Languages.

Learning more about internationalization

There are a variety of online sites that provide general information about internationalization issues. You may find the following Web sites useful if you are learning about internationalization, localization or Unicode. They include:

- Microsoft's Professional Developer's Site for Software Globalization Information (*http://www.microsoft.com/globaldev/default.asp*)
- The definitive word on the W3C's Web site (*http://www.w3.org/international*)
- The Unicode Consortium, a non-profit organization founded to develop, extend and promote use of the Unicode Standard (*http://www.unicode.org*)
- IBM's International Components for Unicode (*http://oss.software.ibm.com/icu/userguide/index.html*)
- A tutorial from Sun on how to internationalize Java applications (*http://java.sun.com/docs/books/tutorial/i18n*)

SilkTest file formats

SilkTest gives you the ability to specify the file format of text files and .ini files. Before SilkTest 5.5, all files were in the ANSI file format. You can create the following formats:

- **ANSI** - For SilkTest purposes, ANSI is defined as the Microsoft Windows ANSI character set from Code Page 1252 Windows Latin 1.
- **Unicode** - Is an extended form of ASCII that provides the ability to represent data in a uniform plaintext format that can be sorted and processed efficiently. Unicode encompasses nearly all characters used in computers today.
- **UTF-8** - (Unicode Transformation Format) Is a multi-byte encoding that can handle all Unicode characters. It is used to compress Unicode, minimize storage consumption and maximize transmission.

You have the ability to save text files in any of three file formats: ANSI, UTF-8, and Unicode. By default all files are saved in UTF-8 format. The Save As dialog boxes throughout include a drop-down list from which you can select the file format in which you want to save your file.

- ANSI files cannot contain non ANSI characters
- The file formats available will depend on the content of your text file. If your file contains characters not available on code page 1252, ANSI will not appear in the drop-down list. If you are working with an existing ANSI file and add non-ANSI characters, the Save As dialog will open when you attempt to save the file. In order to save the changes you will need to change the file format and click Save. For more information see Specifying file formats for existing files.
- The title bar indicates the file format: When you have a file open, the format of that file is indicated on the title bar.
- SilkTest uses the Microsoft standard BOM (Byte Order Marked) to determine the file type for UTF-8 and Unicode files. If a Unicode file does not have the BOM marker then SilkTest sees the file as an ANSI file, and the Unicode characters cannot be displayed.

Working with Bi-directional Languages

SilkTest supports bi-directional languages to the extent that the operating system does. SilkTest captures static text in all Unicode languages. However, scripting, playback and many string functions are not fully supported for complex languages, the most common of these being the bi-directional languages Hebrew and Arabic. The problems you may encounter are discussed below.

SilkTest with bi-directional languages on Windows XP

Windows XP is a multi-lingual operating system and is capable of handling bi-directional languages when configured properly. For additional information, see *Configuring Your Windows XP PC*.

On Windows XP if you input characters from RIGHT to LEFT (CBA) provided that the default system locale is set for a bi-directional language, SilkTest will correctly record and playback the characters as they were entered and appear, from RIGHT to LEFT. When you use a 4Test string function such as StrPos (string position) to return the third element, 4Test correctly counts from right to left and returns `"C"`

Once you have set a default system locale the OS continues to be able read and write that language properly even after another locale has been set as the default. This is provided that the language was never unchecked from the Language Settings area after another default was set. Once a language has been unchecked, when you reboot your system the ability to read and write in that language will be gone. You would need to reset it as the default to restore the capability.

Recording Identifiers for International Applications

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

By default, SilkTest records an object's caption text as its identifier (ID). With SilkTest you can override this default and specify ASCII-only IDs, where the ID is based on the object's class and index. This helps to automatically declare English IDs while keeping the tags native.

The identifier has the form ClassnameIndex, where Classname is the 4Test class of the object and Index is an internally generated integer that ensures that identifiers within a window are unique.

To select ASCII-only identifiers

1. On the Record Window Declarations dialog, select **Options**. The Record Window Declarations Options opens.
2. In the Window declaration identifiers area, select Use the 4Test Class.
3. Click **OK**. ASCII-only IDs will now be captured when you record new window declarations.

Configure Your Environment

Configuring Your Windows XP PC for Unicode content

If you have already configured your Windows XP PC to run your internationalized application, you may be able to disregard this topic and see *Recording identifiers for international applications*.

On Windows XP you may need to do all or some of the following:

- Install language support required by your application through modifications in the Regional and Language Options dialog
- If your application contains content that is in a large character set language, such as simplified Chinese, you may need to install an IME if you want to input data in this language. For more information about IMEs, see the Microsoft support site

Installing Language Support

You must have administrator privileges to install language packs or set the system default locale.

Windows XP provides built-in support for many double-byte languages. Enabling this support can be done at the time of install or after setup through the Regional and Language Options dialog. If you enable language support after setup, you may need files from the Windows XP installation CD. Configurations will vary depending on your needs and how your system has been configured previously. The following instructions are intended only to be general information to get you started.

1. Click **StartSettingsControl PanelRegional and Language Options**.
2. If you are testing East Asian languages, select the **Languages** tab, and then select the **Install files for East Asian languages** check box. You may be prompted to insert the Windows XP CD for the necessary files.
3. Click the **Advanced** tab on the Regional and Language Options dialog, select the language that matches the language of the non-Unicode programs you want to use; for example Chinese (PRC), and then click **OK**.
4. Reboot your computer for the changes to take effect. After you restart your computer, if you want to input data in a language other than the default language, you must click the Language bar icon in your system tray and select the language from the multi-lingual indicator.

If you want to use an Input Method Editor (IME) to input data in the language you selected, you may need to set up your IME.

1. Choose **Start** > **Settings** > **Control Panel** > **Regional and Language Options**.
2. Click the Languages tab and click the **Details** button in the **Text Services and Input Language** area.
3. In the Settings tab on the Text Services and Input Language dialog, select the language you want to use as your default input language.
4. In the Preferences section of the Settings tab, click the Language Bar button, make sure the Show the Language Bar on the desktop check box is selected, and then click OK on the Settings tab. This default will enable your system to display this language in dialogs and menus. We recommend setting the default to the language of the AUT.

Fonts

To display the content of your application in SilkTest you will need to have an appropriate font installed and specify this font in the system registry and in the SilkTest Options/Editor Font. For information about how to configure SilkTest to display double-byte content, see Displaying double-byte characters in SilkTest.

Displaying Double-byte characters

While SilkTest has been widened to process Unicode, displaying double-byte characters is not automatic. Things to keep in mind are:

- Have I configured my operating system to display my content? See *Configuring Your Windows 2000 PC* or *Configuring Your Windows XP PC*.

- Have I configured SilkTest to display double-byte content in its dialogs?
- Do I have the right font set to display my content in the Editor?

Displaying double-byte characters in dialogs

If SilkTest is rendering squares or pipes in dialog boxes where you expect double-byte characters, you may need to make a simple modification to SilkTest using a script we have provided. This script is located in `<SilkTest Installation directory>\Tools`.

1. In SilkTest, choose **File** > **Open**.
2. In the Tools directory, open `font.t`.
3. Click **Run** > **Testcase**. The **Run Testcase** dialog opens.
4. In the arguments area enter, in quotes, the name of the font. For example, `Arial Unicode MS`. It is not necessary to include the type of font, for example `Arial Unicode MS (True Type)`.
5. Click **Run**.
6. Reboot your computer for the changes to take effect.

Displaying double-byte characters in the Editor

In order for the Editor to display double-byte characters, such as those captured in your test frame, you must select a font that is able to display these characters.

1. Within SilkTest, choose **Options** > **Editor Font**.
2. From the available fonts, select one that is able to display the language of your application.

If your application contains multiple languages, make sure that you have a font installed that is capable of rendering all the languages, as the Editor does not display multiple fonts. Licensed Microsoft Office 2000 users can freely download the Arial Unicode MS font from Microsoft.

Localized Browser Support

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest provides support for several internationalized versions of Internet Explorer and Netscape; localized browser include files are provided. For information on what localized browsers are supported, see the Release Notes by clicking **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

SilkTest provides support for several internationalized versions of Internet Explorer and Netscape Navigator. The following files, in ANSI format, are available in the `<SilkTest Installation directory>\Locale`:

- `...\french\browser.inc`
- `...\french\explorer.inc`
- `...\french\netscape.inc`
- `...\german\browser.inc`
- `...\german\explorer.inc`
- `...\german\netscape.inc`

SilkTest also contains the following files in UTF-8 format in the <SilkTest Installation directory>\Locale:

- `...\japanese\browser.inc`
- `...\japanese\explorer.inc`
- `...\japanese \netscape.inc`
- `...\simplified_chinese\browser.inc`
- `...\simplified_chinese\explorer.inc`
- `...\simplified_chinese\netscape.inc`

To change the default English-US browser include files to one of the supported localized browsers:

1. Go to <SilkTest Installation directory>\Locale. In the Locale directory, locate the language of the localized browser you are using.
2. Copy the files contained in the <Language directory> directory.
3. Go to <SilkTest Installation directory>\Extend and paste the files, overwriting the existing files.

To reset support for the English-US browsers to the defaults:

1. Go to <SilkTest Installation directory>\Locale.
2. Copy the files from within the US directory.
3. Go to <SilkTest Installation directory>\Extend and paste the files, overwriting the existing files.
4. Use statements cannot be used to swap browser include files. You must overwrite the files within the Extend directory.

Using an IME with SilkTest

SilkTest supports IMEs with the following limitations:

Windows 2000 (all versions) SilkTest does not record text input with an IME. The following workaround is available:

1. Record the actions and data entry. Where you would normally use the IME to input data, instead use the keyboard to type fictitious data.
2. Click **Done**.
3. Click **Paste to Editor**. Once you have pasted the recorded actions you will be able to modify the input values in the script.
4. To modify the input values, in the script file locate the SetText value that you want to modify. Using the IME, replace the input value with the value you want to input. Be sure to enclose the value in quotes.

Windows 2000 IME support is built-in. Therefore, you can install and use an IME on any version of Windows 2000, not just Windows 2000 localized for Asian languages. The IME is enabled only after you have installed an Asian language package.

For more information about IMEs and for downloads, see the Microsoft support site.

The IME will work once you have installed it, enable it, and are in an application with IME support. In SilkTest, the IME is only available when a file, such as an include or script, is active and additionally on Windows 2000 if a dialog is active.

Work with File Formats

Reusing SilkTest single-byte files as double-byte

If you have existing SilkTest text single-byte files, such as *.pln, *.inc or *.t, that you want to use in double-byte testing, the files must:

- Be compatible with SilkTest, such as files created using the IE 5.x DOM extension for testing a Web application and
- Be recompiled in SilkTest because the object files, *.ino and *.to, are not compatible.

To open an existing SilkTest file as a double-byte file, choose one of the following:

- Copy the file you want to re-use to a new directory. Do not copy the associated object (*.ino or *.to) files. In SilkTest, open this new file, or

- In the existing directory, delete the object files associated with the file you want to re-use. In SilkTest, open the desired file.

When the SilkTest file is compiled, new objects files will be created. If you enter double-byte content into the file, when you try to close the file you will be prompted to save the file in a compatible file format, Unicode or UTF-8.

Specifying file formats for existing files with Unicode content

If you want to save an existing file in a different file format, choose one of the following:

Overwriting the file

If the file is already referenced from other files, you may want to change the format without changing the name or its location. As you cannot have two files with the same name saved in the same directory, even in different formats, the only option is to overwrite the file.

1. Make sure the file is the active window. Choose **File** > **Save As** and select the file from the list.
2. From the **Save as format** drop-down list, select the file format. ANSI is not available if the file contains characters outside of the ANSI character set.
3. Click **Save**. A dialog appears asking if you want to overwrite the file.
4. Click **Yes**.

Saving in the same directory

If you want to have versions of a file in various formats within the same directory, you must save each file with a different name.

1. Make sure the file is the active window. Choose **File** > **Save As**.
2. In the **File name** text box, enter the new name of the file.
3. From the **Save as format** drop-down list, select the file format. ANSI is not available if the file contains characters outside of the ANSI character set.
4. Click **Save**.

Saving in a different directory

If you would like to keep the name of the file but change the format, you must save the file in a different directory.

1. Make sure the file is the active window. Choose **File** > **Save As**. Select the file from the list.
2. Navigate to the directory in which you want to save the file.
3. From the **Save as format** drop-down list, select the file format. ANSI is not available if the file contains characters outside of the ANSI character set.
4. Click **Save**.

If you modify an ANSI text file and the modifications include characters outside of the ANSI characters set, when you try to save your changes, the Save As dialog will open and you need to either overwrite the ANSI file with a file of the same name but in a different format, or rename the file and save in Unicode or UTF-8 format .

Specifying file formats for new files with Unicode content

This topic contains instructions on specifying the file format for:

With the exception of test frames, to specify the file format of a new file:

1. Choose **File** > **New**.
2. On the **New** dialog, select the file type.
3. Click **OK**. The untitled file opens.
4. Choose **File** > **Save As**. The **Save As** dialog opens.

5. Navigate to where you want to store the file and enter the name of the file in the **File name** text box.
6. Select a file format (UTF-8 is the default) from the **Save as format** drop- down list. ANSI is not available if the file contains characters outside of the ANSI character set.
7. Click Save.

To specify the file format for a new test frame:

1. Choose **File** > **New**.
2. On the **New** dialog, select the file type **Test Frame** and click **OK**. The **New Test Frame** dialog opens.
3. To select a file format, click **Browse**. The Save As dialog opens. The default file format for test frames is UTF-8. If you simply type the path and file name in the File name text box of the **New Test Frame** dialog and click **OK**, the file is saved in UTF-8.
4. Navigate to where you want to store the file and enter the name the file in the **File name** text box.
5. Select the file format from the **Save as format** drop-down list. If you select ANSI and if the file contains characters outside of the ANSI character set, when you try to save the file you will need to change the file format to a compatible format, Unicode or UTF-8.
6. Click **Save**. The **New Test Frame** dialog regains focus.
7. On the **New Test Frame** dialog, select the application and proceed as normal.

If you modify an ANSI text file and the modifications include characters outside of the ANSI characters set, when you try to save your changes, the **Save As** dialog will open and you need to either overwrite the ANSI file with a file of the same name but in a different format, or rename the file and save in Unicode or UTF-8 format .

Troubleshooting

Troubleshooting Unicode content

This topic contains helpful information on the following:

Display Issues

Why are my window declarations recording only "pipes"?

What are "pipes" and "squares" anyway?

Why can I only enter "pipes" into a SilkTest frame/include/etc. file?

Why do I see pipes and squares in the Project tab?

Why can't my system dialogs display multiple languages?

Why do I see pipes and squares in my Win32 AUT?

Why do the fonts on my system look so different?

Why don't Unicode characters appear in the SilkTest Project Explorer?

Why isn't my Web application displaying characters properly?

File Formats

Considerations for VB/ActiveX applications

Why am I getting compile errors?

Why does SilkTest open up the Save as dialog when I try to save an existing file?

Working with IMEs

Why don't I see the IME Language bar icon?

Why is English the only language listed when I click the Language bar icon?

Why isn't SilkTest recording my IME input?

Why does this IME look so different from other IMEs I have used?

Display Issues

Why are my window declarations recording only pipes

You've probably forgotten to set the **Options** > **Font Editor** to a font that can display the language of your AUT. See Displaying double-byte characters dialogs.

What are pipes and squares anyway

The pipes and squares (You may also see -?- questions marks) appear in place of the characters because the system has not yet been configured to display these characters. A font that does not support the language is being used in the dialogs and menus. Whether or not you see pipes or squares depends on what font is used and what language you are trying to display.

Why can I only enter pipes into a SilkTest frameincludeetc. file

The SilkTest Editor Font is not set to display the language of your AUT. To learn how to set the Editor Font to a font that can display the language, see Displaying double-byte characters in dialogs.

Why do I see pipes and squares in the Project tab

Pipes and squares (You may also see -?- questions marks) appear in place of characters because the system has not yet been configured to display these characters. A font that does not support the language is being used in the dialogs and menus. Whether or not you see pipes or squares depends on what font is used and what language you are trying to display.

You must configure your system and make sure that you have set the regional settings; see *Configuring Your Windows 2000 PC* or *Configuring Your Windows XP PC* for instructions.

Why cant my system dialogs display multiple languages

If you are testing an application whose content contains multiple languages, meaning that it has several character sets represented, you may need to:

- Make sure that you have a font installed on your machine that can display all the languages.
- Configure SilkTest to use a font that can display your content. For more information, see Displaying double-byte characters in dialogs.

Why do I see pipes and squares in my Win32 AUT

If you start up your application under test and see "pipes" and "squares" in the title bar, menus, or dialog boxes, it may mean that operating system cannot support your application or that your system has not been properly configured to display your content:

On Windows 2000, see *Configuring Your Windows 2000 PC*.

On Windows XP, see *Configuring Your Windows XP PC*.

Why do the fonts on my system look so different

Fonts that appear in your menus, title bars and such are controlled by the Registry settings and the **Display Properties** > **Appearance** settings of your computer.

If your fonts appear too large or too small, you may have incorrectly set the appearance for an item:

1. Go to **Start** > **Settings** > **Control Panel** > **Display.**
2. Go to the **Appearance** tab and select **Windows** standard in the **Scheme** field.
3. Click **OK**.

Your desktop should now appear normal.

Why dont Unicode characters appear in the SilkTest Project Explorer

To view Unicode characters in the SilkTest Project Explorer, you must have installed a language pack with Unicode characters.

See *Why do I see pipes and squares in the Project tab?* for more information.

Why isnt my Web application displaying characters properly

If your web application is not displaying the characters properly, or strange symbols or character are mixed in with your content, you may need to change a setting in your browser.

Internet Explorer Users

Check the settings for Encoding:

1. In Internet Explorer, click **View** > **Encoding**.
2. Select one of the following:
   - From the listed encodings, select one that meets your application's needs.
   - Select **More**, then select an encoding that meets your application's needs.
   - Select **Auto-Select**.

Netscape Users

Check the settings for Character Coding:

1. In Netscape, click **View** > **Character Coding**.
2. Select one of the following:
   - From the listed character coding, select one that meets your application's needs.
   - Select **More**, then select a character coding that meets your application's needs.
   - Select **Auto-detect**.

If you still have problems, you may want to check that your system locale is set for the language of your AUT.

File Formats

Considerations for VBActiveX applications

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If your VB/ActiveX application uses richtextbox controls, you need to base that control on `riched32.dll` version 5.00.213.4.1 which supports Unicode text characters. Before you begin using SilkTest with Unicode content, check to make sure you do not have the older `riched32.dll` version (4.00.993.4). To do this:

1. Using Windows Explorer, locate the `riched32.dll` on your system.
2. Right-click the file and select **Properties**.
3. In the **riched32.dll Properties** dialog, click the **Version** tab. Make sure you have version 5.00.213.4.1.

Why am I getting compile errors

You may be trying to compile a file with an incompatible file format. SilkTest supports three file formats: ANSI, UTF-8, and Unicode. If you try to compile files in SilkTest that are in other formats, such as DBCS, you will get compile errors.

Workaround - In a Unicode-enabled text editor, save the file in one of the SilkTest supported file formats: ANSI, UTF-8 or Unicode.

Why does SilkTest open up the Save as dialog when I try to save an existing file

You have likely added content to the file that is incompatible with the file's existing file format. For example, you could have added Japanese characters to a frame file that was previously saved in ANSI format.

You must save the existing file in a compatible format. For more information, see *Specifying file formats for existing files*.

Working with IMEs

Why dont I see the IME Language bar icon

The Language bar icon only appears if you install the IME. First, make sure that you have an IME on your machine. If you are running:

- **Windows 2000** you do not have to install an IME; Windows 2000 comes with IME support. For information, see *Installing Language Support*.
- **Windows XP** make sure you selected the Show the language bar on the desktop option. For information, see *Configuring Your Windows XP PC*.

Why is English the only language listed when I click the Language bar icon

You must be running an application, or area within the application, that supports an IME for a language other than English to be displayed in the Language bar icon. Applications that support IME include elements of SilkTest such as include files and script files, Outlook, and Internet Explorer.

Why isnt SilkTest recording my IME input

SilkTest cannot record text input with an IME on Windows 2000. A workaround is available. For more information, see *Using an IME*.

Why does this IME look so different from other IMEs I have used

IMEs can look different, depending on the operating system you are using and the particular IME you have accessed. For more information about IMEs, see Microsoft's support site.

Using Autocomplete

Overview of AutoComplete

AutoComplete makes it easier to work with 4Test, significantly reducing scripting errors and decreasing the need to type text into your 4Test files by automatically completing functions, members, application states, and data types. There are four AutoComplete options:

Option	Description
Function Tip	Provides the function signature in a tooltip.
MemberList	Displays window children, properties, methods, and variables available to your 4Test file.
AppStateList	Displays a list of the currently defined application states.
DataTypeList	Displays a list of built-in and user-defined data types.

AutoComplete works with both SilkTest-defined and user-defined 4Test files.

If you create a new 4Test file, you must name and save it as either a .t , .g.t, or .inc file in order for AutoComplete to work. After a 4Test file is saved, AutoComplete recognizes any changes you make to this file in the 4Test Editor and includes files that you reference through a 4Test use statement or the **Use Files** text box on the **Runtime Options** dialog box. When working with an existing 4Test file, you do not need to save or compile in order to access newly defined functions, methods, or members.

AutoComplete only works with 4Test files, which are .t, .g.t, and .inc files, that use hierarchical object recognition or dynamic object recognition with locator keywords.

AutoComplete does not work on comment lines or within plan, suite, or text files. AutoComplete does not support global variables of type window. However, AutoComplete supports Unicode content.

AutoComplete does not distinguish between SilkTest Agents. As a result, AutoComplete displays all methods, properties, variables, and data types regardless of the SilkTest Agent that you are using. For example, if you are using the SilkTest Open Agent, functions and data types that work only with the

SilkTest Classic Agent are also displayed when you use AutoComplete. For details about which methods are supported for each SilkTest Agent, review the corresponding .inc file, such as the `winclass.inc` file.

Customizing your MemberList

The members that you see in the MemberList depend on the MemberList options that you select. You can specify which members appear in your MemberList. The members are window children, methods, properties, and variables. You can also determine how much detail is displayed in the MemberList by specifying the inheritance level and deciding whether you want to view class, data type, and function return type for methods in your MemberList.

All member options are enabled by default and the default inheritance level is below `AnyWin` class, meaning that methods for any class derived from the `AnyWin` class appear in the MemberList. For more information about the inheritance level, see the *AutoComplete Area of General Options Dialog Box*.

Note: Methods that are defined in and above the `AnyWin` class, such as `Click` and `Exist`, which are defined in the Winclass, will not appear in the MemberList. You can type these methods into your script, but they will not appear in the MemberList unless you change the inheritance level to `All`.

To customize your MemberList:

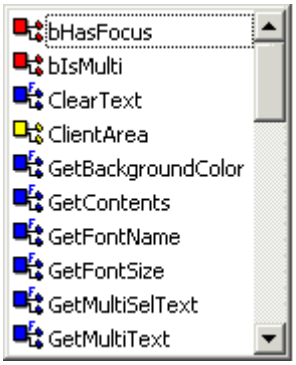
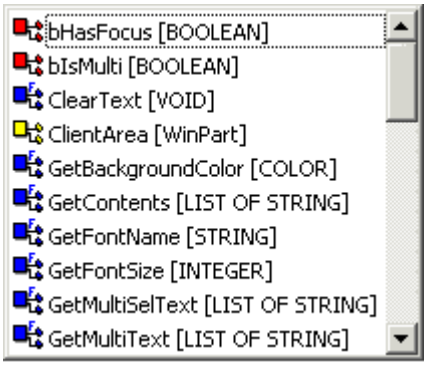
1. Open SilkTest and choose **Options** > **General**.
2. In the **AutoComplete** area of the **General Options** dialog box, make sure MemberList is selected.
3. In the **MemberList Options** area, select the members that you want to display in your MemberList. For example, if you want to view only properties and variables, uncheck the **Methods** and **Window Children** check boxes.
4. Select the appropriate Inheritance Level for the selected methods.

   You can choose one of the following:

Below AnyWin Class	Displays methods for any class derived from the `AnyWin` class. **Below AnyWin Class** is the default.
All	Displays the complete inheritance for members all the way up through `AnyWin` and the control classes, including the `Winclass`.
None	Displays only those members defined in the class of the current object and window declaration.

5. If you want to view attributes for the selected members, such as the class for window children, the data type for properties and variables, and the return type for method functions in your MemberList, check the **Member Type** check box.

   Member Type is not checked by default. The following is a sample MemberList with and without **Member Type** checked.

Default MemberList	MemberList with Member Type Selected
	

6. Click **OK** on the **General Options** dialog box to save your changes.

Frequently Asked Questions about AutoComplete

Why isn’t AutoComplete working?

AutoComplete only works with 4Test files with extension .t, .g.t, and .inc. If (untitled) is displayed in the title bar of your 4Test file, the file has not been saved yet. Save the file as .t, .g.t, or .inc.

After a 4Test file is saved, AutoComplete recognizes any changes you make to this file in the 4Test Editor and include files that you reference through a 4Test use statement or the **Use Files** text box on the **SilkTest Runtime Options** dialog box. Once you save a new file as a .t, .g.t, or .inc, you do not need to save or compile in order to access newly defined functions, methods, or members.

AutoComplete does not work on comment lines or within plan, suite, or text files.

Why doesn't a member appear in my MemberList?

There a few reasons you may not see a member in your MemberList. Here's what you should do:

1. On the **General Options** dialog box, make sure that you chose to show members of this type in the **MemberList Options** section. For additional information, see *Customizing your MemberList*.
2. Make sure the member you want to see is included in the inheritance level you selected. Below `AnyWin` class is the default; you might need to change your inheritance level to `All`. For additional information, see *Customizing your MemberList*.
3. Name and save your file with a .t, .g.t, or .inc extension.
4. Compile your file and fix any scripting errors. Anything following a compile error is not displayed in the MemberList or FunctionTip.

What happens if there is a syntax error in the current file?

Everything, based on the AutoComplete options you have selected, prior to the syntax error will appear in your MemberList and/or FunctionTip. Anything following the syntax error will not appear in your MemberList and/or FunctionTip. For additional information, see *Customizing your MemberList*.

What if I type something and AutoComplete does not find a match?

AutoComplete might not find a match for a number of reasons, for example because of the AutoComplete options you have specified or because of a compile error in your file. For information about fixing some of these issues, see *Customizing your MemberList*, *Turning AutoComplete Options Off* and *Why doesn't a member appear in my MemberList*.

When AutoComplete does not find a match in the MemberList, focus remains on the first item in the list.

Note: If you perform any of the selection methods, which means if you press Return, Tab, or click, the item will be pasted to the Editor.

You can simply type any function, method, or member in your 4Test files; AutoComplete does not restrict you in any way from typing in 4Test files.

Note: You must dismiss the MemberList or FunctionTip before you can type in the Editor.

If you plan to use AutoComplete extensively, we recommend that you rename your identifiers in your window declarations. Knowing your identifier names helps, especially when working with long lists.

Why doesn’t list of record type appear in the FunctionTip?

This is a known limitation. FunctionTip does not support list of record types.

Why does AutoComplete show methods that are not valid for a 4Test class?

When using AutoComplete, the member list occasionally may reveal methods that are not valid for the 4Test class. The compiler will not catch these usage problems, but at Runtime the following exception is raised when the script is played back: `Error: Function <invalid method> is not defined for <window class>.`

Why does AutoComplete show methods, properties, variables, and data types that are not supported for the SilkTest Agent that I am using?

AutoComplete does not distinguish between SilkTest Agents. As a result, AutoComplete displays all methods, properties, variables, and data types regardless of the SilkTest Agent that you are using. For example, if you are using the SilkTest Open Agent, functions and data types that work only with the SilkTest Classic Agent are also displayed when you use AutoComplete. For detailed information about which methods are supported for each SilkTest Agent, review the corresponding .inc file, such as the `winclass.inc` file.

Turning AutoComplete Options Off

This topic contains instructions on how to disable AppStateList, DataTypeList, FunctionTip, and MemberList. For information about customizing your MemberList, see *Customizing your MemberList*.

To turn off AutoComplete options:

1. Open SilkTest and choose **Options** > **General**.
2. In the **AutoComplete** area of the **General Options** dialog box, uncheck the check box for each of the AutoComplete options that you want to disable, and then click **OK**.

Using AppStateList

To display a list of currently defined application states:

1. Within your script, .t or .g.t, or within the include file, type your testcase declaration, followed by the keyword appstate and then press **space**.

   For example `testcase foo ( ) appstate` .

   A list of currently defined application states displays. You can also type the keyword basedon followed by a **space**. For example `appstate MyAppState () basedon` .
2. Use one of the following methods to select the appropriate member and paste it to the Editor.
   - Type the first letter or the first few letters of the member and then press **Enter** or **Tab**.
   - Use your arrow keys to locate the member and then press **Enter** or **Tab**.
   - Scroll through the list and click on a member to select it.

Using DataTypeList

To display a list of built-in and user-defined data types:

1. Within your script, .t or .g.t, or include file, type `array` or `varargs`, as appropriate, followed by the of keyword and a **space**.

   For example, `list of`.

   The current list of built-in and user-defined data types appears. You can also view the list of data types by pressing **F11**.
2. Use one of the following methods to select the appropriate member and paste it to the Editor:
   - Type the first letter or the first few letters of the member and then press **Enter** or **Tab**.
   - Use your arrow keys to locate the member and then press **Enter** or **Tab**.

- Scroll through the list and click on a member to select it.

Using FunctionTip

To display the function signature for a function, testcase, or method.

1. Within your script, .t or .g.t, or include file, type the function, testcase, or method name, followed by an open parenthesis " (".

   for example `SetUpMachine(`. The function signature displays in a tooltip with the first argument, if any, in bold text. The function signature includes the return argument type, pass-mode, data type, name of the argument, and null and optional attributes, as they are defined.
2. Type the argument.

   The FunctionTip containing the function signature remains on top and highlights the argument you are expected to enter in bold text. As you enter each argument and then type a comma, the next argument that you are expected to type is highlighted. The expected argument is always indicated with bold text; if you backspace or delete an argument within your function, the expected argument is updated accordingly in the FunctionTip. The FunctionTip disappears when you type the close parenthesis ") " to complete the function call.

If you want to dismiss the FunctionTip, press **Escape**. FunctionTip is enabled by default. See *Turning AutoComplete Options Off* if you want to disable FunctionTip.

Using MemberList

This topic contains instructions on how to use MemberList to view and select a list of members.

To view a list of members:

1. Customize the member list so that it displays the information you require.

   You can choose to display any or all of the following members:

Member	Description
Window children	Displays all window objects of type WINDOW that are defined in window declarations in the referenced .t, .g.t, and .inc files. Indicated in the MemberList with a yellow icon.
Methods	Displays all methods defined in the referenced .t, .g.t, and .inc files. Indicated in the MemberList with a blue icon.
Properties	Displays all properties defined in the referenced .t, .g.t, and .inc files. Indicated in the MemberList with a red icon.
Variables	Displays all defined variables in the referenced .t, .g.t, and .inc files, including native data types, data, and records. Fields defined for records and nested records also display in the list. Indicated in the MemberList with a red icon.

2. Within your script or include file, type the member name and then type a period (.).

   For example `Find.`.

   The MemberList displays. Depending on the MemberList Options and the Inheritance Level you select, the types of members that display in the MemberList will vary.
3. Use one of the following methods to select the appropriate member and paste it to the Editor:
   - Type the first letter or the first few letters of the member and then press **Enter** or **Tab**.
   - Use your arrow keys to locate the member and then press **Enter** or **Tab**.
   - Scroll through the list and click on a member to select it.

The MemberList is case sensitive. If you type the correct case of the member, it is automatically highlighted in the MemberList; press **Enter** or **Tab** once to paste it to the Editor. If you do not type the correct case, the

member has focus, but is not highlighted; press **Enter** or **Tab** twice to select the member and paste it to the Editor. To dismiss the MemberList, press **Escape**.

Using the Extension Kit

Overview of the Extension Kit

This functionality is available only for projects or scripts that use the Classic Agent.

The SilkTest Extension Kit (Extension Kit) enables you to write new 4Test Agent functions in C. By writing a 4Test Agent function in C, you can call your function from your 4Test scripts, which your function runs on the target machine, and your function can interact with the Agent. The Extension Kit provides nearly limitless testing power.

You must install the Extension Kit when you install SilkTest. For details on using the Extension Kit, refer to the *Extension Kit Guide for .NET* or the *Extension Kit Guide for Windows*, in `Start/Programs/Silk/SilkTest <version>/Documentation/SilkTest Classic/Extension Kit Guide for .NET` or `Start/Programs/Silk/SilkTest <version>/Documentation/SilkTest Classic/Extension Kit Guide for Windows`.

The following topics provide answers to the following frequently asked questions about the Extension Kit:

- What is the Extension Kit (EK)? What can it do?
- How do I determine if the Extension Kit is a viable option?
- How does the Extension Kit relate to my application?
- How much time and work is required to write extension functions?
- How portable are extension functions?
- How do extension functions differ from built-in functions?
- My custom objects can't be easily handled by 4Test alone. Should I use DLL-calling or the Extension Kit?
- What modifications need to be made to the application under test?
- What skills are required in order to use the Extension Kit?

What is the Extension Kit What can it do

This functionality is available only for projects or scripts that use the Classic Agent.

The Extension Kit is an Application Programming Interface (API) to SilkTest. It is not an executable application. It is a header file which exposes the functionality of a library that already ships with SilkTest. It is one of the available alternatives for handling custom objects from within SilkTest.

The Extension Kit is simple to use and immensely powerful. It enables you to:

- Access some of the internal functionality of SilkTest and extend the functionality of SilkTest in a way that can be virtually transparent to the scripter.
- Pass information from SilkTest to your C code and back again.
- Perform keyboard and mouse actions using the capabilities of the Agent.

What Skills are Required in Order to use the Extension Kit

This functionality is available only for projects or scripts that use the Classic Agent.

The following skills are required to use the Extension Kit:

- C programming experience.

- Experience with the operating system and Windows.
- SilkTest experience.
- An understanding of the internal functioning of the application under test or an understanding of how to use its API.

How much Time and Work is Required to Write Extension Functions

This functionality is available only for projects or scripts that use the Classic Agent.

The amount of time and work that is required to write extension functions depends on many factors, among which the following:

- Experience and prowess of the person developing the extension functions.
- Does the application already have an appropriate (and documented) API?
- Are the custom objects really objects or are they drawings?

How do I Determine if the Extension Kit is a Viable Option

This functionality is available only for projects or scripts that use the Classic Agent.

Use the Extension Kit when you need to provide custom GUI support for complex applications by writing custom functions, or by adding new classes or methods to existing classes. To determine if the Extension Kit is a viable option, decide if you can perform the task in C. If you can perform the task in C, you can do it with an extension function.

When determining if the Extension Kit is an option, consider the following:

Did you write the objects yourself?	If so, you are virtually unlimited in what you can do, since you have access to all the information.
Are the objects written by a third-party vendor?	If the objects have a well-documented API that supports the functionality you need, you can use the Extension Kit. If you are using third-party objects with no API, or an insufficient one, the Extension Kit is not usable from a technical standpoint.

The objects do not need to be visible to SilkTest for it to be able to manipulate them or retrieve information. For example, if the objects are not visible to the Recorders in the first place, the Extension Kit will not change that. You can set up declarations that will allow you to write truly object-oriented scripts against controls that are not objects at all.

How Portable are Extension Functions

This functionality is available only for projects or scripts that use the Classic Agent.

Internal extension functions are largely portable across platforms. People who are familiar with writing multi-platform code should have no difficulty.

Youve determined that custom objects cant be easily handled by 4test alone. Should you use DLL-calling or the Ex

This functionality is available only for projects or scripts that use the Classic Agent.

If you only need to do a few simple things, it might be in your best interest to use DLL-calling. If you're doing anything extensive, it's probably worthwhile to use the Extension Kit, for the following reasons:

- Functions written with the Extension Kit can be significantly faster than calling equivalent DLL functions.
- All 4Test Data types are supported, so there is no need to convert between 4Test and C data types.
- Extension Kit functions are really methods of a specific class.
- You can use the event-generation capabilities of the Agent.
- You can create functions which take optional arguments.
- You can create functions which raise 4Test exceptions.
- The Extension Kit is available on all platforms, while DLL-calling is only available on PC-class platforms.

How does the Extension Kit Relate to my Application

This functionality is available only for projects or scripts that use the Classic Agent.

The Extension Kit interface consists of C functions. However, most applications, for which an extension is to be written, are in C++. You can write extension functions in C++ and create C wrapper functions for them. However, it would be handy to have a C++ class for the Extension Kit interface functions.

The Extension Kit interfaces with `assist.dll`. The version of `assist.dll` must match the version of SilkTest. Extension writers can either link `assist.lib` into the application under test (AUT), or dynamically load `assist.dll`. If `assist.lib` is statically linked, then the AUT requires `assist.dll` to reside on the user's system. In that case, testers either must use a version of the AUT that differs from the shipped version, or the large `assist.dll` must be shipped with the AUT. Therefore it is preferable to dynamically load `assist.dll`. To dynamically load `assist.dll`, you need to add code to the AUT to load the DLL and create pointers to the DLL functions that are used by the extension.

You can create a C++ class that wraps the Extension Kit functions. These functions can also dynamically load `assist.dll` and create pointers to the `assist.dll` functions. For additional information about the class, refer to the *Extension Kit Guide for .NET* or the *Extension Kit Guide for Windows*. The class is briefly documented in *The Framework of an Extension Function* in Chapter 3.

The class is implemented as two files, `FWxQapDynDll.cpp` and `FWxQapDynDll.h`. When you install the Extension Kit, these files are copied to `<SilkTest installation directory>/EKWIN32/Examples`.

The original files only allow extension functions to accept zero or one parameter. For functions requiring more than 5 parameters, you can modify the files yourself. For additional information, refer to the prototypes and implementations for *RegisterClassFun* in the .cpp and .h files. Follow the template in these headers. To obtain the files, contact *Technical Support*. The .h file can be used as is. The .cpp file requires modification, since you must add references to your application-specific extension functions. Search for `TODO` in the .cpp file in order to customize it for your extension.

Note: The first part of `FWxQapDynDll.h` imports `qapwinek.h`, so you do not need to include `qapwinek.h` in your application code.

What Modifications need to be Made to the Application Under Test

This functionality is available only for projects or scripts that use the Classic Agent.

You may not have to make modifications to the application under test (AUT). If it is a PC-class platform, you can choose to write an external extension. If the application already supports the needed messages, you won't need to make changes. Modifications to the application itself should be relatively minor. Extension functions need to retrieve data, like text values, and occasionally call pre-existing functions.

How do Extension Functions Differ from Built-In Functions

This functionality is available only for projects or scripts that use the Classic Agent.

Use the Extension Kit in conjunction with built-in SilkTest features to get nearly seamlessly added functionality.

- Class mapping
- Winclass definitions

You can write scripts as if the functions were built in.

- Built-in functions display under **Show Methods** in the **Record Actions** and **Record Identifiers** windows.
- Extension functions display in the **Library Browser**, and you can fully document them there if you want to. The Extension Kit does not change the recorders. **Record Actions** does not show the extension functions that you define.

Using the Library Browser

Overview of the Library Browser

Click **Help** > **Library Browser** to access the **Library Browser**. It provides online documentation for:

- Built-in 4Test methods, properties, and functions: the **Library Browser** shows the name and class of the method, one line of descriptive text, syntax, and a list of parameters, including a description.
- User-defined methods: the **Library Browser** shows the name and class of the method, syntax, and a list of parameters. It displays User defined as the method description and displays the data type for each parameter.
- User-defined Properties: As with user-defined methods, the description for user-defined properties by default is User defined.

The **Library Browser** does not, by default, provide documentation for your user-defined functions. You can add to the contents of the Library Browser to provide descriptive text for your user-defined methods, properties, and functions.

Library Browser source file

The core contents of the Library Browser are based on a standard SilkTest text file, `4test.txt`, which contains information for the built-in methods, properties, and functions.

You can edit `4test.txt` to include your user-defined information, or define your site-specific information in one or more separate files, and then have SilkTest compile the file (creating `4test.hlp`) to make it available to the Library Browser. (Information about methods in `4test.hlp` is also used in the **Verify Window** dialog for methods.) See *Adding to the Library Browser* and *Using multiple source files in the Library Browser*.

SilkTest does not update `4test.txt` with user-defined information; instead it populates the **Library Browser** from information it receives when include files are compiled in memory. You modify `4test.txt` to override the default information displayed for user-defined objects.

Simply looking through `4test.txt` should give you all the help you need about how to structure the information in the file. The following table lists all the keywords and describes how they are used in 4test.txt. You should edit a copy of `4test.txt` to add the information you want.

If you need help, see *Example: documenting user-defined information in the Library Browser*.

Keywords

Keywords are followed by a colon and one or more spaces.

class	Name of the class.
function	Name of the function. Specify the full syntax. If the function returns a value, specify: `return_value = function_name (parameters)` Otherwise, specify: `function_name (parameters)`
group	Name of the function category.
method	Description of the method. Specify the full syntax. If the method returns a value, specify: `return_value = method_name (parameters)` Otherwise, specify: `method_name (parameters)`
notes	Description of the method, property, or function, up to 240 characters. Do not split the description into multiple **notes** fields, since only the first one is displayed.
parameter	Name and description of a method or function parameter. Each parameter is listed on its own line. Specify the name, followed by a colon, followed by the description of the parameter.
property	Name of the property.
returns	Type and description of the return value of the method or function. Specify the name, followed by a colon, followed by the description of the return value.
#	Comment.

Adding Information to the Library Browser

1. Make a backup copy of the default `4test.txt file`, which is in the directory where you installed SilkTest, and store your backup copy in a different directory.
2. In an ASCII text editor, open `4test.txt` in your SilkTest installation directory and edit the file. See examples for methods, properties, and functions, if necessary.
3. Quit SilkTest.
4. Place your modified `4test.txt` file in the SilkTest installation directory.
5. Restart SilkTest. SilkTest sees that your source file is more recent than `4test.hlp` and automatically compiles `4test.txt`, creating an updated `4test.hlp`. If there are errors, SilkTest opens a window listing them and continues to use the previous `4test.hlp` file for the Library Browser. If there were errors, fix them in `4test.txt` and restart SilkTest. Your new definitions are displayed in the **Library Browser** (assuming that the files containing the declarations for your custom classes, methods, properties, and functions are loaded in memory).

There is another approach to updating the **Library Browser**: maintain information in different source files. See *Using multiple source files in the Library Browser*.

If the **Library Browser** isn't displaying your user-defined objects, close the **Library Browser**, recompile the include files that contain your user-defined objects, then reopen the **Library Browser**.

Have the SilkTest add user-defined files to the Library Browser

1. Create a text file that includes information for all your custom classes and functions, using the formats described in the *Library Browser source file*.
   If you have added methods or properties to built-in classes, you should add that information in the appropriate places in 4test.txt, as described above. Only document your custom classes and functions in your own help file.
2. Choose **Options** > **General** and add your help file to the list in the **Help Files For Library Browser** field. Separate the files in this list with commas.
3. Click **OK**. SilkTest recompiles `4test.hlp` to include the information in all the files listed in the **Help Files For Library Browser** field. If there are errors, SilkTest opens a window listing them and continues to use the previous `4test.hlp` file for the **Library Browser**. If you had errors, fix them in your source file, then quit and restart SilkTest. SilkTest recompiles `4test.hlp` using your modified source file.

Viewing functions in the Library Browser

To view information about built-in 4Test functions in the **Library Browser**:

1. Choose **Help** > **Library Browser**, and then click the **Functions** tab.
2. Select the category of functions you want in the **Groups** list box. To see all built-in 4Test functions, select the **Include all** check box. Functions are listed in the **Functions** list box.
3. Select the function for which want information.

Viewing methods for a class in the Library Browser

4Test classes have methods and properties. When you select the **Methods** or **Properties** tabs in the **Library Browser**, you see a list of all the built-in and user-defined classes in hierarchical form.

To see the methods or properties for a class:

1. Choose **Help** > **Library Browser**, and then click the **Methods** or **Properties** tab.
2. Select the class in the **Classes** list box. Double-click a + box to expand the hierarchy. Double-click a – box to collapse the hierarchy. The methods or properties for the selected class are displayed. By default, only those methods or properties that are defined by the class are displayed. To see all methods or properties that are available to the class (that is, methods or properties also defined by an ancestor of the class), select the **Include inherited** check box. To see all methods or properties (even those not available to the selected class), select the **Include all** check box.
3. Select a method or property. Information about the selected method or property is displayed.

If the **Library Browser** isn't displaying your user-defined objects, close the **Library Browser**, recompile the include files that contain your user-defined objects (**Run** > **Compile**), and then re-open the **Library Browser**.

Examples of documenting user-defined methods

This topic contains examples of adding user-defined methods, properties, and functions to the **Library Browser**. For instructions on how to add information to the **Library Browser**, see *Adding to the **Library Browser*** and *Using multiple source files in the **Library Browser***.

```
#*****************************************************
class:     DialogBox
...
#** custom method
```

```
method:    VerifyNumChild (iExpectedNum)
parameter: iExpectedNum: The expected number of child objects (INTEGER).
notes:     Verifies the number of child objects in a dialog box.

Documenting user-defined properties: Add the property descriptions to the
appropriate class section in 4test.txt, such as:
#*********************************************
class:    DialogBox
...

#** custom property
property: iNumChild
notes:    The number of child objects in the dialog box.

Documenting user-defined functions: Create a group called User-defined
functions and document your functions, such as:
group:     User-defined functions

function:  FileOpen (sFileName)
parameter: sFileName = "myFile": The name of the file to open.
notes:     Opens a file from the application.

function:  FileSave (sFileName)
parameter: sFileName = "myFile": The name of the file to save.
notes:     Saves a file from the application.
```

Web classes not displayed in Library Browser

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

The class hierarchy in the **Library Browser** does not include the web classes (`BrowserChild`, `HtmlText`, and so on).

Possible Causes and Solutions

No browser extension is enabled.	Make sure that at least one browser extension is enabled.
Enhanced support for Visual Basic is enabled.	Disable Visual Basic by un-checking the **ActiveX** check box for the Visual Basic application in the **Extension Enabler** and **Extensions** dialog boxes.

Using Source Control Software

Overview of SCCI support

SilkTest supports source control application integration via the Source Code Control Interface (SCCI).

SilkTest allows you to work with most source control applications that are SCCI compliant such as:

- Visual Source Safe
- Concurrent Versions System (CVS)
- Perforce SCM System

SilkTest supports source code control provider implementations of SCCI version 1.1. Check the Release Notes for the most up-to-date information about supported versions, any known issues, and workarounds.

Note: SilkTest no longer supports StarTeam or PVCS Version Manager.

When you start SilkTest, it detects, loads, and initializes your source control SCCI library. If this fails, there is no source control support. If support is activated, the SilkTest menu bar is updated to include:

- Two new menu items under **File**: **Check Out** and **Check In**.
- One new menu item under **Options**: **Source Control Options**. This is a cascading menu with the choices **Archive Setup**, **Label Versions**, **View History Log**, and **View Differences**.

These new menu items allow you to directly communicate with your source control application; the dialogs that appear are actually part of that application.

Prerequisites and recommendations

You must be familiar with your source control application before you begin using it via SilkTest. If you are not, consult your vendor's documentation for directions.

Perforce is a transactional-based system and therefore acts slightly differently than SourceSafe. Be sure to review the Perforce documentation before beginning to use it with SilkTest.

The first time you access your source control application after starting SilkTest, you may be prompted to log on, depending on how your site has implemented security. There is no source control security set up through SilkTest. If you do not know your user logon and password, consult your system administrator.

SilkTest lets you access dialogs and prompts from your source control application from within SilkTest. If you are not sure how to respond to the dialogs that appear, you can get more information by pressing F1 or clicking the **Help** button.

We strongly recommend that you use your source control application and not SilkTest to create new archives or projects. Your source control application offers a full set of options where the SCC interface offers a smaller number of options.

General steps in using source control software

The way you use source control in SilkTest is determined by your source control application. These instructions assume that you are already familiar with how that application works; if you have any questions about how to respond to the dialogs that appear, see your system administrator.

The actual process is determined by your source control application, but the general steps in using source control software with SilkTest are:

1. Select a 'project' archive and working directory by using Archive Setup dialog* from the **Check Out** or **Check In** dialogs.
2. Get the current copy of an archived file by using the **Check Out** dialog*.
3. Check out the most recent copy of the archived file by using the **Check Out** dialog*.
4. Make changes to the local file, then check it in by using the **Check Out** dialog*.

   Optionally, add new files to source by using the **Advanced Add** button on the **Check In** dialog

* The dialog that appears is generated by your source control application. Refer to your source control application documentation for more information.

We strongly recommend that you use your source control application and not SilkTest to create new archives or projects. Your source control application offers a full set of options where the SCC interface offers a smaller number of options.

Setting up source control

SilkTest does not install a source control application. You must purchase and install such applications separately. During installation SilkTest reads and uses the window registry keys of your default source control application. If you have more than one source control application installed on your machine, the

default is listed in your Registry under the following key: `HKEY_LOCALMACHINE\Software\SourceCodeControlProvider\ProviderRegKey`.

Typically, an application is set as the default source control during its installation, however, some vendors provide a mechanism to set their application as the default source control provider. Refer to your source control application's documentation for instructions.

If you do not have a source control application installed or if SilkTest was unable to find and load the SCCI implementation library, the menu items for **File** > **Check Out**, **File** > **Check In**, and **Optioins** > **Source Control Options** are not available.

Running Tests and Interpreting Results

Running Tests

Recording a Testcase With the Classic Agent

When you record a testcase with the Classic Agent, SilkTest uses hierarchical object recognition, a fast, easy method to create scripts. However, testcases that use dynamic object recognition are more robust and easy to maintain. You can create tests for both dynamic and hierarchical object recognition in your test environment. Use the method best suited to meet your test requirements. You can use both recognition methods within a single testcase if necessary.

1. Enable extensions and set up the recovery system.
2. Click **Record Testcase** on the Basic Workflow bar. If the workflow bar is not visible, click **Workflows** > **Basic** to enable it.
3. Type the name of your testcase in the Testcase name field of the **Record Testcase** dialog. Testcase names are not case sensitive; they can be any length and consist of any combination of alphabetic characters, numerals, and underscore characters.
4. Select **DefaultBaseState** in the **Application State** field to have the built-in recovery system restore the default BaseState before the testcase begins executing. If you chose **DefaultBaseState** as the application state, the testcase is recorded in the script file as: `testcase testcase_name ()`. If you chose another application state, the testcase is recorded as: `testcase testcase_name () appstate appstate_name`.
5. If you do not want SilkTest to display the status window it normally shows during playback when driving the application to the specified base state—perhaps because the status bar obscures a critical control in the application you are testing—uncheck the **Show AppState status window** check box.
6. Click **Start Recording**. SilkTest:
   - Closes the **Record Testcase** dialog.
   - Starts your application, if it was not already running.
   - Removes the editor window from the display.
   - Displays the **Record Status** window.
   - Waits for you to take further action
7. Interact with your application, driving it to the state that you want to test. As you interact with your application, SilkTest records your interactions in the **Testcase Code** field of the **Record Testcase** dialog, which is not visible.
8. To review what you have recorded, click the **Done** button in the **Record Status** window.

   SilkTest displays the **Record Testcase** dialog, which contains the 4Test code that has been recorded for you.
9. To resume recording your interactions, click the **Resume Recording** button in the dialog. To temporarily suspend recording, click the **Pause Recording** button on the **Record Status** window.
10. Verify the testcase.

Creating a suite

After you have created a number of script files, you might want to collect them into a test suite. A suite is a file that names any number of scripts. Instead of running each script individually, you run the suite, which executes in turn each of your scripts and all the testcases they contain. Suite files have a `.s` extension.

1. Click **File** > **New**.
2. Select the **Suite** radio button and click **OK**. SilkTest displays an untitled suite file.
3. Enter the names of the script files in the order you want them executed. For example, the following suite file executes the `find.t` script first, the `goto.t` script second, and the `open.t` script third:

   ```
   find.t
   goto.t
   open.t
   ```
4. Click **File** > **Save** to save the file.
5. If you are working within a project, SilkTest prompts you to add the file to the project. Click **Yes** if you want to add the file to the open project, or **No** if you do not want to add this file to the project.

Passing arguments to a script

You can pass arguments to a script. For example, you might want to pass in the number of iterations to perform or the name of a data file. All functions and testcases in the script have access to the arguments.

How to pass arguments to a script

All arguments are passed in as strings, separated by spaces, such as: Bob Emily Craig

If an argument is more than one word, enclose it with quotation marks. For example, the following passes in three arguments: `"Bob H" "Emily M" "Craig J"`

You can pass arguments to a script using the following methods:

- Specify them in the **Arguments** field in the **Runtime Options** dialog (**Options** > **Runtime** from the menu bar).
- The **Arguments** field in the **Run Testcase** dialog is used to pass arguments to a testcase, not to an entire script.
- Specify them in a suite file after a script name, such as: `find.t arg1 arg2`
- Provide arguments when you invoke SilkTest from the command line.
- If you pass arguments in the command line, the arguments provided in the command line are used and any arguments specified in the currently loaded options set are not used. To use the arguments in the currently loaded options set, do not specify arguments in the command line.

Processing arguments passed into a test script

You use the `GetArgs` function to process arguments passed into a script. `GetArgs` returns a list of strings with each string being one of the passed arguments. Any testcase or function in a script can call `GetArgs` to access the arguments.

Example: passed arguments

The following testcase prints a list of all the passed arguments:

```
testcase ProcessArgs ( )
LIST OF STRING lsArgs
lsArgs = GetArgs ( )
ListPrint (lsArgs)

//You can also process the arguments individually. The following testcase
```

```
prints the second argument passed:
testcase ProcessSecondArg ( )
LIST OF STRING lsArgs
lsArgs = GetArgs ( )
Print (lsArgs[2])

//The following testcase adds the first two arguments:
testcase AddArgs ()
LIST OF STRING lsArgs
lsArgs = GetArgs ( )
NUMBER nArgSum

nArgSum = Val (lsArgs[1]) + Val (lsArgs[2])
Print (nArgSum)
```

You can use the `Val` function to convert the arguments (which are always passed as strings) into numbers.

The `Val` function was used to specifying arguments `10 20 30` results in the following:

```
Script scr_args.t (10, 20, 30) - Passed
Passed:  1 test (100%)
Failed:  0 tests (0%)
Totals:  1 test, 0 errors, 0 warnings

Testcase AddArgs - Passed

30
```

Running a testcase

When you run a testcase, SilkTest interacts with the application by executing all the actions you specified in the testcase and testing whether all the features of the application performed as expected.

SilkTest always saves the suite, script, or testplan before running it if you made any changes to it since the last time you saved it. By default, SilkTest also saves all other open modified files whenever you run a script, suite, or testplan. To prevent this automatic saving of other open modified files, uncheck the **Save Files Before Running** check box in the **General Options** dialog.

1. Make sure that the testcase you want to run is in the active window.
2. Click **Run Testcase** on the **Basic Workflow** bar. If the workflow bar is not visible, choose **Workflows** > **Basic** to enable it.
3. SilkTest displays the **Run Testcase** dialog, which lists all the testcases contained in the current script.
4. Select a testcase and specify arguments, if necessary, in the **Arguments** field. Remember to separate multiple arguments with commas.
5. To wait one second after each interaction with the application under test is executed, check the **Animated Run Mode (Slow-Motion)** check box. Typically, you will only use this check box if you want to watch the testcase run. For instance, if you want to demonstrate a testcase to someone else, you might want to check this check box. Executions of the default base state and functions that include one of the following strings are not delayed:
   - Verify
   - Exists
   - Is
   - Get
   - Set
   - Print
   - ForceActiveXEnum
   - Wait
   - Sleep

6. To view results using the **Silk TrueLog Explorer**, check the **Enable TrueLog (for Classic Agent only)** check box. Click **TrueLog Options** to set the options you want to record.

   The **Silk TrueLog Explorer** works with the SilkTest Classic Agent only. Use the **Difference Viewer** to analyze results for testcases that use the SilkTest Open Agent.
7. Click **Run**. SilkTest runs the testcase and generates a results file.

   For the SilkTest Classic Agent, multiple tags are supported by Windows 2000 and Windows XP Agents. If you are running testcases using other Agents, you can run scripts that use declarations with multiple tags. To do this, check the **Disable Multiple Tag Feature** check box in the **Agent Options** dialog on the Compatibility tab. When you turn off multiple-tag support, 4Test discards all segments of a multiple tag except the first one.

Running a testplan

Before running a testplan, make sure that the window declarations file for the testplan is correctly specified in the **Runtime Options** dialog and that the testplan is in the active window.

- To run the entire testplan, choose **Run** > **Run All Tests**. SilkTest runs each testcase in the plan and generates a results file.
- To run only tests that are marked, choose **Run** > **Run Marked Tests**. SilkTest runs each marked test and generates a results file.

You can also run a single testcase without marking it. See the Testplan menu.

If your testplan is structured as a master plan and associated subplans, SilkTest automatically opens any closed subplans before running. For more information on using subplans, see *Dividing a testplan into a master plan and subplans*. SilkTest always saves the suite, script, or testplan before running it if you made any changes to it since the last time you saved it. By default, SilkTest also saves all other open modified files whenever you run a script, suite, or testplan. To prevent this automatic saving of other open modified files, uncheck the **Save Files Before Running** check box in the **General Options** dialog.

Running the currently active script or suite

1. Make sure the script or suite you want to run is in the active window.
2. Choose **Run** > **Run**. SilkTest runs all the testcases in the script or suite and generates a results file.

Stopping a Running Testcase Before it Completes

To stop running a testcase before it completes:

- If your test application is on a target machine other than the host machine, choose **Run** > **Abort**.
- If your test application is running on your host machine, press `Shift+Shift`.

Setting a Testcase to Use Animation Mode (Slow-Motion)

To slow down a testcase during playback so that it can be observed, set the testcase to use animation mode. For instance, if you want to demonstrate a testcase to someone else, you might want to use animation mode.

You can specify the animation mode when you run a testcase, or you can specify the animation mode in the **Runtime Options** dialog.

To specify the animation mode using the Runtime Options dialog

1. From the main menu, choose **Options** > **Runtime**.
2. In the **Runtime Options** dialog, check the **Animated Run Mode (Slow-Motion)** check box.

3. Click **OK**.

Interpreting Results

Overview of the results file

A results file provides information about the execution of the testcase, script, suite, or testplan. By default, the results file has the same name as the executed script, suite, or testplan, but with a `.res` extension (for example, `find.res`).

Whenever you run tests, SilkTest generates a results file, which indicates how many tests passed and how many failed, describes why tests failed, and provides summary information. You can invoke comparison tools from within the results file that pinpoint exactly how the runtime results differ from your known baselines. Testplan results files offer additional features, such as the ability to generate a Pass/Fail report or compare different runs of the testplan. When SilkTest displays a results file, on the menu bar it includes the Results menu, which allows you to manipulate the results file and locate errors. The Results menu appears only when the active window displays a results file.

Silk TrueLog Explorer

SilkTest also provides the **Silk TrueLog Explorer** to help you analyze test results files. You must configure SilkTest to use the **Silk TrueLog Explorer** and specify what you want to capture. For more information about using **Silk TrueLog Explorer**, see the **Silk TrueLog Explorer** User Guide for SilkTest.

Multiple User Environments

A `.res` file can be opened by multiple users, as long as no test is in process. This means you cannot have two users run tests at the same time and write to the same results file. You can run a test on the machine while the file is open on the other machine. However, you must not add comments to the file on the other machine, or you will corrupt the .res file and will not be able to report the results of the test. If you add comments to the file on both machines, the comments will be saved only for the file that is closed (and therefore saved) first.

Default Settings

By default, the results file displays an overall summary at the top of the file, including the name of the script, suite, or testplan; the machine the tests were run on; the number of tests run; the number of errors and warnings; actual errors; and timing information. To hide the overall summary, click the summary and click **Results** > **Hide Summary**. For a script or suite results file, the individual test summaries contain timing information and errors or warnings. For a testplan results file, the individual test summaries contain the same information as in the overall summary plus the name of the testcase and script file.

While SilkTest displays the most current version of the script, suite, or testplan, by default SilkTest saves the last five sets of results for each script, suite, or testplan executed. (To change the default number, use the **Runtime Options** dialog.) As results files grow after repeated testing, a lot of unused space can accumulate in the files. You can reduce a results file's size with the Compact menu option.

The format for the rest of a testplan results file follows the hierarchy of test descriptions that were present in the testplan. Test statements in the testplan that are preceded by a pound sign (#) as well as comments (using the `comment` statement) are also printed in the results file, in context with the test descriptions.

To change the default name and directory of the results file, edit the **Runtime Options** dialog.

Note: If you provide a local or remote path when you specify the name of a Results file in the **Directory/Field** field on the **Runtime Options** dialog, the path cannot be validated until script execution time.

Viewing test results

Whenever you run tests, SilkTest generates a results file, which indicates how many tests passed and how many failed, describes why tests failed, and provides summary information.

1. Click the **Explore Results** button on the **Basic Workflow** or the **Data Driven Workflow** bars.
2. On the **Results Files** dialog, navigate to the file name that you want to review and click **Open**.

By default, the results file has the same name as the executed script, suite, or testplan. To review a file in the **SilkTest TrueLog Explorer**, open a `.xlg` file. To review a SilkTest results file in SilkTest, open a `.res` file.

The difference viewer

To evaluate application logic errors, use the **Difference Viewer**, which you can invoke by clicking the box icon following an error message relating to an application's behavior.

Some expanded error messages are preceded by a box icon and three asterisks. What happens when you click the box icon depends on the error message.

If the error message relates to an application's:

- Appearance, as in Bitmaps have different sizes, SilkTest opens the bitmap tool for your platform. (The bitmap tool compares baseline and results bitmaps.)
- Behavior, as in Verify selected text failed, SilkTest opens the **Difference Viewer**. (The **Difference Viewer** compares actual and expected values for a given testcase. It lists every expected (baseline) value in the left pane and the corresponding actual value in the right pane. Differences are marked with red, blue, or green lines, which denote different types of differences, for example, deleted, changed, and added items.)

You can use **Results** > **Next Result Difference** to find the next difference and update the values using **Results** > **Update Expected Value**.

Errors and the Results file

You can expand the text of an error message or have SilkTest find the error messages for you. To navigate from a testplan test description in a results file to the actual test in the testplan, click the test description and select **Results** > **Goto Source**.

Navigating to errors in the script

SilkTest provides several ways to move from the results file to the actual error in the script:

- Double-click in the margin next to an error line to go to the script file that contains the 4Test statement that failed.
- Click an error message and select **Results** > **Goto Source**.
- Click an error message and press **Enter**.

What the box icon means

Some expanded error messages are preceded by a box icon and three asterisks.

If the error message relates to an application's behavior, as in `Verify selected text failed`, SilkTest opens the **Difference Viewer**. (The **Difference Viewer** compares actual and expected values for a given testcase.)

Application appearance errors

When you click a box icon followed by a bitmap-related error message, SilkTest starts the bitmap tool, reads in the baseline and result bitmaps, and opens a **Differences** window and **Zoom** window.

Bitmap tool

In the **Bitmap Tool**:

- The Baseline Bitmap is the bitmap that SilkTest expected (the baseline for comparison)
- The Results Bitmap is the actual bitmap that SilkTest captured
- The Differences window shows the differences between the baseline and result bitmap

The bitmap tool supports several comparison commands, which let you closely inspect the differences between the baseline and results bitmaps.

Finding application logic errors

To evaluate application logic errors, SilkTest provides the **Difference Viewer**, which you can invoke by clicking the box icon following an error message relating to an application's behavior.

The Difference viewer

Clicking the box icon invokes the **Difference Viewer**'s double-pane display-only window. It lists every expected (baseline) value in the left pane and the corresponding actual value in the right pane.

SilkTest highlights all occurrences where expected and actual values differ. On color monitors, differences are marked with red, blue, or green lines, which denote different types of differences, for example, deleted, changed, and added items.

When you have more than one screen of values or are using a black-and-white monitor, use **Results** > **Next Result Difference** to find the next difference. Use **Update Expected Values**, described next, to resolve the differences.

Updating expected values

You might notice upon inspecting the **Difference Viewer** or an error message in a results file that the expected values are not correct. For example, when the caption of a dialog changes and you forget to update a script that verifies that caption, SilkTest will log errors when you run the testcase. To have your testcase run cleanly the next time, you can modify the expected values with the **Update Expected Value** command. Note that this command updates data within a testcase, not data passed in from the testplan.

Debugging tools

You might need to use the debugger to explore and fix errors in your script. In the debugger, you can use the special commands available on the **Breakpoint**, **Debug**, and **View** menus.

Marking failed testcases

When a testplan results file shows testcase failures, you might choose to fix and then rerun them one at a time. You might also choose to rerun the failed testcases at a slower pace, without debugging them, simply to watch their execution more carefully.

To have SilkTest identify the failed testcases, make the results file active and select **Results** > **Mark Failures in Plan**. SilkTest marks all the failed testcases and makes the testplan the active file.

Testplan Pass Fail Report and Chart

A **Pass/Fail** report lists the number and percentage of tests that have passed during a given execution of the testplan. The report can be subtotaled by an attribute, for example, by Developer.

After you generate a **Pass/Fail** report, you can take these actions:

- Print the report.
- Export the report to a comma-delimited ASCII file.
- Chart a generated Pass/Fail report—that is, produce report information as a graph—or you can directly graph the testplan results information without a preexisting report.

You can mark manual tests as having passed or failed in the **Update Manual Tests** dialog. The **Pass/Fail** report includes in its statistics the manual tests that you have documented as having passed or failed.

Overview of merging testplan results

Results files consist of a series of results sets—one set for each testplan run. You can merge different results sets in a results file. Merging results sets is useful when:

- Sections of the testplan are run separately (either by one person or by several people) and you need to create a single report on the testing process. That is, you want one results set that includes the different runs.
- The testplan is updated with new tests or subplans and you want a single results set to reflect the execution of the additional tests or subplans.

SilkTest combines the two results sets by merging the results set you selected in the **Merge Results** dialog into the currently open results set. The open results set is altered. No additional results set is created. The date and time of the altered results set reflect the more recent test run.

For example, let's say that yesterday you ran a section of the testplan consisting of 20 tests and today you ran a different section of the testplan consisting of 10 tests. The merged results set would have today's date and would consist of the results of 30 tests.

Analyzing Results with the Silk Truelog Explorer

Silk TrueLog Explorer

The Silk TrueLog Explorer helps you analyze test results files and can capture screenshots before and after each action, and when an error occurs.

The Silk TrueLog Explorer works with the SilkTest Classic Agent only. Use the Difference Viewer to analyze results for testcases that use the SilkTest Open Agent.

You can turn TrueLog on or off:

- For all testcases using the TrueLog options dialog.
- Each time you run a specific testcase using the Run Testcase dialog.
- At runtime using the test script.

When you turn TrueLog on or off in the Run Testcase dialog, SilkTest makes the same change in the TrueLog Options dialog. Likewise, when you turn TrueLog on or off in the TrueLog Options dialog, SilkTest makes the same change in the Run Testcase dialog.

For more information about Silk TrueLog Explorer, choose Start/Programs/Silk/SilkTest <version>/ Documentation/SilkTest Classic/Silk TrueLog Explorer User Guide.

Opening the TrueLog Options dialog

Use the TrueLog options to enable the Silk TrueLog Explorer and to customize the test result information that the Silk TrueLog collects for SilkTest.

The Silk TrueLog Explorer works with the SilkTest Classic Agent only. Use the Difference Viewer to analyze results for testcases that use the SilkTest Open Agent.

To open the Silk TrueLog Explorer Options dialog from the main menu

- Click **Options** > **TrueLog**

To open the Silk TrueLog Explorer Options dialog from a testcase

1. Click **Run Testcase** on the **Basic Workflow** bar. If the workflow bar is not visible, choose **Workflows** > **Basic** to enable it.
2. In the **Run Testcase** dialog, check the **Enable TrueLog (for Classic Agent only)** check box and then click **TrueLog Options**.

Setting Silk TrueLog Explorer Options

Use the TrueLog options to enable the Silk TrueLog Explorer and to customize the test result information that the Silk TrueLog collects for SilkTest.

The Silk TrueLog Explorer works with the SilkTest Classic Agent only. Use the Difference Viewer to analyze results for testcases that use the SilkTest Open Agent.

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

1. Open the **TrueLog Options** dialog.
2. To capture TrueLog data and activate logging settings, check the **Enable TrueLog (for Classic Agent only)** check box and then choose to capture data for:
   - **All Testcases** logs activity for all testcases, both successful and failed. This setting may result in large TrueLog files.
   - **Testcases with errors** logs activity for only those testcases with errors. This is the recommended setting, as it limits the size of TrueLog files.
3. In the **TrueLog File** field, specify the location and name of the TrueLog file.

   This path is relative to the machine on which the SilkTest Agent is running. The name defaults to the name used for the results file, with an `.xlg` extension. The location defaults to the same folder as the testcase `.res` file.

   Note: If you provide a local or remote path in this field, the path cannot be validated until script execution time.
4. To set pre-determined logging levels, in the TrueLog Presets section, choose one of the following:
   - **Minimal logs testcases with errors**; enables bitmap capture of desktop on error; does not log any actions.
   - **Default logs testcases with errors**; enables bitmap capture of window on error; logs data for Select and SetText actions; enables bitmap capture for Select and SetText actions.
   - **Full logs all testcases**; logs all control information; logs all events for browsers except for MouseMove events; enables bitmap capture of the window on error; captures bitmaps for all actions.

   If you enable Full logs and encounter a `Window Not Found` error, you may need to manually edit your script.
5. In the **Log the following for controls** section, specify the types of information about the controls on the active window or page to log.
6. In the **Log the following for browsers** section, specify the browser events that you want to capture.
7. In the **TrueLog Delay** field, specify the amount of time you want to allow Windows to draw the application window before a bitmap is taken. The delay can be used for browser testing. You can insert

a `Browser.WaitForReady` call in your script to ensure that the DocumentComplete events are seen and processed. (If WindowActive nodes are missing from the TrueLog, you need to add a `Browser.WaitForReady` call.) You can also use TrueLog Delay to optimize script performance. Set the delay as small as possible to get the correct behavior and have the smallest impact on script execution time. The default setting is 0.

8. To capture screen shots of the application under test, check the **Enable Bitmap Capture** check box and then choose to capture bitmaps:
9. Click the **Action Settings** tab to select the scripted actions you want to include in the TrueLog. When enabled, these actions appear as nodes in the Tree List view of the Silk TrueLog Explorer.
10. In the **Select Actions to Log** section, check the **Enable** check box to include the corresponding a 4Test method action in the log. Each action corresponds to a 4Test method, except for Click and Select.
11. In the **Select Actions to Log** section, from the **Bitmap** list box, select the point in time that you want bitmaps to be captured.
12. Click **OK**.

Turning TrueLog On or Off at Runtime Using a Script

Turn the Silk TrueLog Explorer on and off at runtime to analyze test results, capture screenshots before and after each action, and capture screenshots when an error occurs.

Use the test script to turn TrueLog on or off multiple times during the execution of a testcase. For example, if you run a single testcase to test multiple user interface menus, you can turn TrueLog on and off several times during the script to capture bitmaps for only a portion of the menus.

The Silk TrueLog Explorer works with the SilkTest Classic *Agent* only. Use the Difference Viewer to analyze results for testcases that use the SilkTest Open Agent.

1. Set the Silk TrueLog Explorer options to define what you want the TrueLog Explorer to capture.
2. Create or open the script that you want to modify.
3. Navigate to the portion of the script that you want to turn on or off.
4. To turn TrueLog off, type: `SetOption(OPT_PAUSE_TRUELOG, TRUE)`.
5. To turn TrueLog on, type: `SetOption(OPT_PAUSE_TRUELOG, FALSE)`.
6. Choose **File** > **Save** to save the script.

Viewing results using the Silk TrueLog Explorer

Use the Silk TrueLog Explorer to analyze test results files, capture screenshots before and after each action, and capture screenshots upon error.

The Silk TrueLog Explorer works with the SilkTest Classic Agent only. Use the Difference Viewer to analyze results for testcases that use the SilkTest Open Agent.

1. Set Silk TrueLog Explorer options.
2. Run a testcase.
3. Choose one of the following:
   - Choose **Results** > **Launch TrueLog Explorer**.
   - Click the **Explore Results** button on the **Basic Workflow** or the **Data Driven Workflow** bars.
4. On the Results Files dialog, navigate to the file name that you want to review and click **Open**.

By default, the results file has the same name as the executed script, suite, or testplan. To review a file in the SilkTest TrueLog Explorer, open a `.xlg` file. To review a SilkTest results file in SilkTest, open a `.res` file.

For more information about Silk TrueLog Explorer, click **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Documentation** > **SilkTest Classic** > **Silk TrueLog Explorer User Guide**.

Modifying Your Script to Resolve Window Not Found Exceptions When Using TrueLog

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When you run a script and get a `Window 'name' was not found` error, you can modify your script to resolve the issue. Use this procedure if all of the following options are set in the **TrueLog Options** dialog:

- The action **PressKeys** is enabled.
- Bitmaps are captured after or before and after the **PressKeys** action.
- **PressKeys** actions are logged.

The preceding settings are set by default if you select Full as the TrueLog preset.

To resolve this error, in your testcase, use `FlushEvents()` after a `PressKeys()` and `ReleaseKeys()` pair. Or, you can use `TypeKeys()` instead.

There is no need to add `sleep()` calls in the script or to change timeouts.

```
[-] testcase one()
[ ] Browser.SetActive()
[ ] // Google.PressKeys("<ALT-T>")
[ ] // Google.ReleaseKeys("<ALT-T>")
[ ] Google.TypeKeys("<ALT-T>")
[ ] Agent.FlushEvents ()
[ ] Google.TypeKeys("O")
[ ] Agent.FlushEvents ()
[ ]
[ ] //recording
[ ] IE_Options.SetActive ()
[ ] IE_Options.PageList.Select ("Security")
[ ] IE_Options.Security.SecurityLevelIndicator.SetPosition (2)
[ ] BrowserMessage.SetActive ()
[ ] BrowserMessage.OK.Click ()
[ ] IE_Options.SetActive ()
[ ] IE_Options.OK.Click()
```

Analyzing Bitmaps

Overview of the Bitmap Tool

This topic contains a brief overview of the **Bitmap Tool**. To access more information about the **Bitmap Tool**, launch it and press `F1` or choose **Help** > **Help Topics**.

You can start **Bitmap Tool** from:

- The tool's icon.
- The **Run** dialog.
- SIlkTest results file.

The **Bitmap Tool** is an application that allows you to test and correct your Windows application's appearance by comparing two or more bitmaps and identifying the differences between them. It is especially useful for testing inherently graphical applications, like drawing programs, but you can also check the graphical elements of other applications. For example, you might want to compare the fonts you expect to see in a dialog with the fonts actually displayed, or you might want to verify that the pictures in toolbar buttons have not changed.

It can be used as a stand-alone product, in which you create and compare bitmaps of entire windows, client areas, the desktop, or selected areas of the screen. More commonly, however, you use the tool in conjunction with SilkTest. Bitmaps captured within SilkTest can be opened in the **Bitmap Tool**, where you can compare them using the tool's special comparison features. Conversely, bitmaps captured by the bitmap tool can be compared by SilkTest's bitmap functions.

You can compare a baseline bitmap captured in the **Bitmap Tool** with one captured in a SilkTest testcase of your application.

- If you write testcases by hand, you can use SilkTest's built-in bitmap functions.
- If you prefer to record testcases via **Record** > **Testcase**, the **Verify Window** dialog allows you to record a bitmap-related verification statement.

The **Bitmap Tool** can only recognize an operating system's native windows. In the case of the Abstract Windowing Toolkit (AWT), included with Sun Microsystems Java Development Kit (JDK), each control has its own window, since AWT controls are native Microsoft windows. As a result, the **Bitmap Tool** will only see the top level dialog window.

When to use the Bitmap Tool

You might want to use the **Bitmap Tool** in these situations:

- To compare a baseline bitmap against a bitmap generated during testing.
- To compare two bitmaps from a failed test.

For example, suppose during your first round of testing you create a bitmap using one of SilkTest's built-in bitmap functions, `CaptureBItmap`. Assume that a second round of testing generates another bitmap, which your test script compares to the first. If the testcase fails, SilkTest raises an exception but cannot specifically identify the ways in which the two images differ. At this point, you can open the Bitmap Tool from the results file to inspect both bitmaps.

Overview of Capturing Bitmaps with the Windows Bitmap Tool

You can capture bitmaps by embedding bitmap functions and methods in a testcase or by using the **Bitmap Tool**. This topic explains how to capture bitmaps in the Bitmap Tool.

Use the **Capture** menu to capture a bitmap for any of the following in your application:

- A window.
- The client area of a window (working area, without borders or controls).
- A selected rectangular area of the screen (this is especially useful for capturing controls within a window).
- The desktop.

Overview of Comparing bitmaps

The **Bitmap Tool** can create and graphically locate the differences between two bitmaps. You can use all Windows functionality to resize, save, and otherwise manipulate bitmaps, in addition to the special comparison features included in the tool.

Using Bitmap Tool, you can:

- Show the areas of difference.
- Zoom in on the differences.
- Jump from one zoomed difference to the next.
- View on-line statistics about the bitmaps.
- Edit (copy and paste), print, and save bitmaps.

- Create masks.

The Bitmap Tool has four major comparison commands: **Show**, **Zoom**, **Scan**, and **Comparison Statistics**. You can also compare bitmaps by creating and applying masks.

Overview of SilkTests bitmap functions

`CaptureBitmap`, `SYS_CompareBitmap`, `WaitBitmap`, and `VerifyBitmap` are built-in bitmap-related 4Test functions. In particular, `VerifyBitmap` is useful for comparing a screen image during the execution of a testcase to a baseline bitmap created in the bitmap tool. If the comparison fails, SilkTest saves the actual bitmap in a file. In the following example, the code compares the testcase bitmap (the baseline) against the bitmap of `TestApp` captured by `VerifyBitmap`:

```
TestApp.VerifyBitmap ("c:\sample\testbase.bmp")
```

Baseline and Result bitmaps

To compare two bitmaps, you must designate one bitmap in the comparison as the baseline and the second bitmap as the result. While you may have many bitmap files open in the tool, at any one time only one bitmap can be set as the baseline and one as the result. If you want to set new baseline and result bitmaps, you must first unset the current assignments.

These designations are temporary; you can at any point set and reset a bitmap as a baseline, result, or neither.

Rules for using comparison commands

You should be familiar with these rules before using the commands:

- If you are comparing two new bitmaps captured in the tool, designate one bitmap as the baseline, the other as the result bitmap.
- If you are comparing two existing, saved bitmaps, open first the bitmap that you consider the baseline. The tool automatically designates the first bitmap you open as the baseline, and the second as the result.
- The commands must be used in this order: **Show**, **Zoom**, and **Scan**.

Saving captured bitmaps

You can, if you want, save the bitmaps you have captured in the **Bitmap Tool**. You should adopt a naming convention that helps you distinguish between the first bitmap in the comparison, called the baseline bitmap, and the second bitmap, called the result bitmap. You can make the distinction in the file name itself, for example, by appending or prefixing a "b" or "r" to the name and using the same file extension for all bitmap files. Or you might use the same file name for both baseline and result bitmaps and add a unique file extension.

Example

You save baseline and result bitmaps of the Open dialog as `open.bmp` and `open.rmp`. Alternatively, you might name them `openbase.bmp` and `openres.bmp`, respectively.

The following table lists the file extensions supported by the **Bitmap Tool**. We recommend that you use `.bmp` for baseline bitmaps and `.rmp` for result bitmaps.

If you are saving	And you want the file name to be	Then use this extension
Baseline bitmap	Identical to the result bitmap's	`.bmp`
Result bitmap	Identical to the baseline bitmap's	`.rmp`

If you are saving	And you want the file name to be	Then use this extension
Either baseline or result bitmap	Unique	`.bmp` or `.dib` (Device Independent Bitmap)

SilkTest uses `.rmp` for bitmaps that are captured within a testcase and fail verification.

Capturing a bitmap with the Windows Bitmap Tool

1. Start the application in which you want to capture bitmaps and set up the window or area to capture.
2. Start the **Bitmap Tool**.
3. If you want to change the current behavior of the tool window, click **Capture** > **Hide Window on Capture**. By default, the tool window is hidden during capture.
4. Choose a window or screen area to capture:

Window	Choose **Capture** > **Window**. Click the window you want to capture.
Client area	Choose **Capture** > **Client Area**. Click the client area you want to capture.
Selected rectangular area	Choose **Capture** > **Rectangle**. 1. Move the mouse cursor to desired location to begin capture. 2. While pressing and holding the left mouse button, drag the mouse to outline a rectangle, and then release the mouse button to capture it. During outlining, the size of the rectangle is shown in pixels.
Desktop	Choose **Capture** > **Desktop**.

   The **Bitmap Tool** creates a new MDI child window containing the newly captured bitmap. The title bar reads **Bitmap - (Untitled)** and the status line at the bottom right of the window gives the dimensions of the bitmap (height by width), and the number of colors.
5. Repeat steps 3 and 4 to capture another bitmap. Alternatively, open an existing bitmap file.
6. Save the bitmap.

   Now you are ready to compare the two bitmaps or create a mask for the baseline bitmap.

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that `VerifyBitmap` functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Capturing a bitmap during recording

1. Open the dialog by pointing at the object you want to capture and pressing `Ctrl+Alt`.
2. Click the **Bitmap** tab.
3. Enter a file name in the **Bitmap File Name** field. Use the **Browse** button to select a directory name.

   The default path is based on the current directory. The default file name for the first bitmap is `bitmap.bmp`. Click **Browse** if you need help choosing a new path or name.
4. Choose whether to copy the **Entire Window**, **Client Area of Window**, or **Portion of Window**, and click **OK**.

To capture a portion of the window, move the mouse cursor to the location where you want to begin. While pressing the left mouse button, drag the mouse to outline a rectangle, and then release the mouse button to capture the bitmap.

SilkTest always adds a bitmap footer to the bitmap file. This means that the physical size of the bitmap will be slightly bigger than if you capture the bitmap in the Bitmap Tool. The bitmap footer always contains the window tag for a given bitmap.

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Capturing all or part of the Zoom window while in the scan mode

1. Make sure the **Capture** > **Hide Window** is unchecked. If necessary, select the item to remove the check mark.
2. Choose **Next** or **Previous** until the **Zoom** window contains the difference you want to capture.
3. Press one of the following:

Entire Zoom window	Press `Ctrl+W` and select the **Zoom** window.
Client area of Zoom window	Press `Ctrl+A` and select the **Zoom** window.
Selected area of Zoom window	Press `Ctrl+R`. Move the mouse cursor to desired location to begin capture. While pressing and holding the left mouse button, drag the mouse to the screen location to end capture, and release the mouse button.

4. Optionally, you can fit the bitmap in its window, resize it, and save it.

Unsetting a designated bitmap

Uncheck the menu item. For example, to un-set a baseline bitmap, uncheck **Bitmap** > **Set Baseline**. The check mark is removed.

Zooming the Baseline Bitmap Result Bitmap and Differences window

Choose **Differences** > **Show** and then **Differences** > **Zoom**.

The tool arranges the **Baseline Bitmap** on top, the **Result Bitmap** on the bottom, and the **Differences** window in the middle. To the right of these, the tool creates a **Zoom** window with three panes, arranged like the bitmap windows

Designating a bitmap as a baseline

Click **Bitmap** > **Set Baseline**.

The **Set Baseline** menu item is checked. The title bar of the child window changes to `Baseline Bitmap -- filename.bmp`.

Designating a bitmap as a results file

Click **Bitmap** > **Set Result**.

The **Set Result** menu item is checked. The title bar of the child window changes to `Result Bitmap --filename.rmp`.

Looking at statistics

The **Comparison Statistics** command displays information about the baseline and result bitmaps, with respect to width, height, colors, bits per pixel, number of pixels, and the number and percentage of differences (in pixels).

Viewing statistics comparing the baseline bitmap and result bitmaps

Click **Differences** > **Comparison Statistics**.

The **Bitmap Comparison Statistics** window appears.

The number of colors is derived from the following formula: number of colors = 2 ^ (bits per pixel).

Exiting from scan mode

Click **Differences** > **Scan**.

Exiting scan leaves the tool in zoom mode.

Starting the Bitmap Tool

You can start Bitmap Tool from:

- The tool's icon.
- The **Run** dialog.
- SilkTest results file.

Starting the Windows Bitmap Tool from its icon and open bitmap files

1. Click the **Bitmap Tool** icon in the SilkTest program group.
   The **Bitmap Tool** window appears.
2. Do one of the following:

Open an existing bitmap file	Choose **File** > **Open** and specify a file in the **Open** dialog. See *Comparing bitmaps*.
Capture a new bitmap	See *Capturing bitmaps*.

Starting the Windows Bitmap Tool from the Results File

When the verification of a bitmap fails in a testcase, SilkTest saves the actual result in a bitmap file with the same name as the baseline bitmap but with the extension `.rmp`. So, if the bitmap file `testbase.bmp` fails the comparison, SilkTest names the result bitmap file `testbase.rmp`. It also logs an error message in the results file.

In some cases this error message does not reflect an actual error. In particular, when SilkTest compares a bitmap it captured with one captured in the bitmap tool, the comparison fails because SilkTest stores footer information in its bitmap. The bitmaps might in fact be identical in all ways except for this information.

To compare the actual bitmap generated by the testcase against the baseline bitmap generated by the bitmap tool or one of SilkTest's built-in functions, click the box icon preceding the error message.

SilkTest opens the bitmap tool, opens both baseline (expected, `.bmp` file) and result (actual, `.rmp` file) bitmaps, creates a **Results/View Differences** and places it in between the baseline and result bitmaps. To the right, the tool displays a three-paned Zoom window.

Starting the Bitmap Tool from the Run dialog

1. Click **Start** > **Run**. The **Run** dialog appears.
2. Type the pathname of the tool's executable file and any parameters in the Command Line text field and click **OK**. The **Bitmap Tool** starts. Any bitmaps you specified on the command line are opened. Go to Comparing bitmaps. If you did not specify any files in the command line, go to the next step. You can now open existing bitmaps created in SilkTest or in the tool, or you can capture new bitmaps:

Open an existing bitmap file	Choose **File** > **Open** and specify a file in the **Open** dialog. See *Comparing bitmaps*.
Capture new bitmaps	See *Capturing bitmaps*.

Using Masks

A mask is a bitmap that you apply to the baseline and result bitmaps in order to exclude any part of a bitmap from comparison by the bitmap tool. For example, if you are testing a custom object that is painted on the screen and one part of the object is variable, you might want to create a mask to filter out the variable part from the bitmap comparison.

You might consider masking any differences that you decide are insignificant or that you know will vary in an effort to avoid testcase failure. For example, suppose a testcase fails because one bitmap includes a flashing area of a dialog. In the bitmap tool you can block the flashing area from the two bitmaps by creating and applying a mask to them. Once a mask is applied and the masked bitmaps are saved, the mask becomes a permanent part of the baseline bitmaps you are comparing. Masks can also be saved in separate files and used in SilkTest testcases.

You can create a mask in two ways:

- By converting the **Differences** window to a mask. A mask created this way filters out all differences.
- By opening a new mask window and specifying rectangular areas to mask.

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Prerequisites for the masking feature

Before using the masking feature, you must:

- Capture or open two bitmaps to compare. Set `baselinesetbaseline` and `resultsetresult` bitmaps, if currently unset.
- Determine which sections you need to mask. (Use one or more comparison `featurescomparisoncmds`, if necessary, to locate bitmap differences.)

Applying a mask

1. Open the mask bitmap file, if not open, and choose **Bitmap** > **Set Mask**.

2. Choose **Edit** > **Apply Mask**.

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Editing an applied mask

You can edit a mask after it has been applied:

- To add to the mask, place the mouse cursor in the baseline bitmap window at the position where you want to begin adding to the mask. As you press and hold the left mouse button, drag the mouse cursor to outline a rectangle. Then release the left mouse button.
- To delete part of the mask, place the mouse cursor in the baseline bitmap window at the position where you want to begin deleting part of the mask. While pressing and holding the Shift key, drag the mouse cursor over the area of the existing map that you want to delete, and then release the Shift key and the left mouse button.

Creating a mask that excludes some differences or just selected areas and applying it

1. Click **Edit** > **New Mask**. The bitmap tool creates an empty **Mask Bitmap child** window that is the same size as the baseline bitmap.
2. Using the **Differences** window to help you locate differences, place the mouse cursor in the baseline bitmap window at the position where you want to begin creating the mask. As you press and hold the left mouse button, drag the mouse cursor to outline a rectangle. Then release the left mouse button. The rectangular outline in the baseline map changes to a filled-in rectangle. The mask bitmap window also contains a like-sized rectangle in the same relative location.
3. Repeat step the previous step until you have completed the mask.
4. If you want to delete a portion of the mask, place the mouse cursor in the baseline bitmap window at the position where you want to begin editing. While pressing the Shift key and then the left mouse button, drag the mouse cursor over the area of the existing map that you want to delete, and then release the Shift key and the left mouse button.

   The area of the mask overlapped by the rectangle outline disappears in both the baseline and mask bitmap window.
5. Choose **Edit** > **Apply Mask**. The bitmap tool applies the mask to the result bitmap and closes the **Differences** window.
6. Choose one of the following actions:

Keep the baseline and result bitmaps with the mask applied	Save the bitmap files. The mask is now a permanent part of the bitmap files.
Unapply the mask	Close the mask bitmap window. Saving is optional.
Keep the mask as it is	Save the mask file.
Edit the mask	Choose **File** > **Save** and close the mask bitmap window (which un-applies the mask).

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color

depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Creating a mask that excludes all differences and applying it

1. Open a Differences window, if one is not already open, click **Differences** > **Show**.
2. Click **Differences** > **Convert to Mask**.
   A message is displayed: `Bitmaps are now identical on screen.`
3. Click **OK**.

   The bitmap tool creates an untitled mask bitmap from the **Differences** window, swapping black and white, and applies the mask to the baseline and result bitmaps.
4. Choose one of the following actions:

Keep the baseline and result bitmaps with the mask applied	Save the bitmap files. The mask is now a permanent part of the bitmap files.
Unapply the mask	Close the mask bitmap window. Saving is optional.
Keep the mask as it is	Save the mask file.
Edit the mask	Choose **File** > **Save** and close the mask bitmap window (which un-applies the mask).

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

Saving a mask

Masks can be saved in a file, applied to the baseline and result bitmaps for you to examine on screen only, or applied to and saved in the baseline and result bitmap files. Once masks are applied and saved, they become a permanent part of the baseline and result bitmaps. The advantage of saving the mask alone is that later you can read in the mask file and apply it to the bitmap on screen, thus allowing you to keep the bitmap in its original state.

You can supply the name of a mask bitmap file (as well as its associated baseline bitmap file) as an argument to SilkTest's bitmap functions.

The bitmap tool supports the `.msk` file extension for mask files. Alternatively, you can designate a mask in the file name and use the generic `.bmp` extension. We recommend, however, that you use the `.msk` extension.

The following SilkTest bitmap-related functions accept mask files as arguments:

- `GetBItmapCRC`
- `SYS_CompareBItmap`
- `VerifyBItmap`
- `WaitBItmap`

The Open Agent and Classic Agent capture bitmaps in different color depths. By default, the Open Agent captures bitmaps using the current desktop settings for color depth. The Classic Agent captures bitmaps as 24-bit images. If you create a mask file for the captured bitmap using the Bitmap tool that comes with SilkTest, the mask file is saved as a 24-bit bitmap. If the bitmap mask file does not have the same color depth as the bitmap that you capture, an error occurs. To ensure that VerifyBitmap functions do not fail due

to different color-depth settings between the captured image and the mask image, ensure that the bitmaps have the same color depth.

For Differences

Scanning bitmap differences

Click **Differences** > **Scan** or **Differences** > **Next**.

The tool indicates the location of the first difference it finds by placing a square in the same relative location of the **Baseline**, **Result**, and **Differences** windows. The three panes of the **Zoom** window also show the difference.

Showing areas of difference

The **Show** command creates a **Differences** window, which is a child window containing a black-and-white bitmap. Black represents areas with no differences and white represents areas with differences.

Graphically show areas of difference between a baseline and a result bitmap

Click **Differences** > **Show**.

The bitmap tool displays a **Differences** window along with the source baseline and result bitmaps from which it was derived.

Jumping from one difference to the next

The **Scan** command on the **Differences** menu automates zoom mode and causes the bitmap tool to scan for differences from left to right and top to bottom. When the first difference is found, a small square (32 x 32 pixels) is shown in the **Baseline Bitmap**, **Result Bitmap**, and **Differences bitmap** windows in the same relative location. In addition, that location is shown in all three panes in the **Zoom** window. Use **Differences** > **Next** to jump forward and **Differences** > **Previous** to jump back.

You must first create a **Differences** window and a **Zoom** window using **Differences** > **Show** and **Differences** > **Zoom**.

Moving to the next or previous difference

Click **Differences** > **Next** or **Differences** > **Previous**.

Zooming in on the differences

The **Zoom** command creates a special (not sizable) **Zoom** window with three panes and resizes and stacks the **Baseline**, **Differences**, and **Result** windows.

- The top pane of the **Zoom** window contains a zoomed portion of the **Baseline Bitmap** window.
- The middle pane shows a zoomed portion of the **Differences** window.
- The bottom pane shows a zoomed portion of the **Result Bitmap** window.

All three zoomed portions show the same part of the bitmap. When you move the mouse within any of the three windows, the bitmap tool generates a simultaneous and synchronized real-time display in all three panes of the **Zoom** window.

While in scan mode, you can capture the **Zoom** window to examine a specific bitmap difference.

Working with Results Files

Attaching a comment to a result set

You can attach comments to individual results sets to record useful information about the test run:

1. Open the results file.
2. Choose **Results** > **Select** to display the **Select Results** dialog.
3. Select the results set to which you want to attach a comment.
4. Type the comment in the **Comment** text field at the bottom of the dialog. The comment appears in the Comment column in the **Select Results** dialog.
5. Click **OK**.

SilkTest displays the comments in the various dialogs that list results sets, such as the **Extract Results** and **Delete Results** dialogs.

Comparing results files

The **Compare Two Results** command allows you to quickly note only the results that have changed from a prior run without having to look at the same errors over again. The command identifies differences based on the following criteria:

- A test passes in one testplan run and fails in the other.
- A test fails in both runs but the error is different.
- A test is executed in one testplan run but not in the other.

SilkTest uses the test descriptions as well as the test statements to identify and locate the various cases in the testplan. Therefore, if you change the descriptions or statements between runs, SilkTest will not be able to find the test when you run **Compare Two Results**.

1. Open two results files.
2. Make the results set you want to compare to another results set the active window.
3. Choose **Results** > **Compare Two Results**.
4. On the **Compare Two Results** dialog, select a results set from the list box and click **OK**.
5. When SilkTest displays the results set again, it positions a colored arrow in the left margin for every test that is different.

   A red arrow indicates that the difference is due to the pass/fail state of the test changing.

   A magenta arrow indicates that the difference is due to the addition or removal of the test in the compared test run.
6. Click **Results** > **Next Result Difference** to search for the next difference or choose **Results** > **Next Error Difference** to search for the next difference that is due to the change in a pass/fail state of a test.

SilkTest uses the test descriptions as well as the script, testcase, and testdata statements to identify and locate the various cases in the testplan and in the results set. When test results overlap in the two results set that were merged, the more recent run is used. If you change a test description between runs or modify the statements, SilkTest might be unable to find the test when you try to merge results. SilkTest places these orphaned tests at the top of the results set.

Customizing results

You can modify the way that results appear in the results file as follows:

- Change the colors of elements in the results file
- Change the default number of results sets
- Display a different set of results
- Remove the unused space in a results file

You can also view an individual summary.

Deleting a results set

1. Click **Results** > **Delete**. SilkTest displays the **Delete Results** dialog with the most current results set displayed first.
2. Select the set of results you want to delete and click **OK**.

Change the default number of results sets

1. Choose **Options** > **Runtime**. SilkTest displays the **Runtime Options** dialog.
2. In the **History Size** field, change the number to the number of results files you want.
3. By default, SilkTest keeps five results sets.

Changing the colors of elements in the results file

1. In SilkTest, click **Options** > **Editor Colors** to display the **Editor Colors** dialog.
2. Select an element from the **Editor Item** list box.
3. Select one of the 16 colors from the palette or modify the RGB values of the selected color. To modify RGB value, select the color. Slide the bar to the left or right, click the spin buttons, or type specific RGB values until you get the color you want.
4. When you are satisfied with the color, click **OK**.

To revert to the default colors, click **Reset**. By default, these results file elements are displayed in the following colors:

Results file element	Default color/icon
Error messages and warnings	Red plus sign (bold on black-and-white monitor)
Warnings only	Purple plus sign
Test descriptions of executed tests	Dark blue
Test descriptions of unexecuted tests	Grayed out
Other descriptive lines	Black

Fix incorrect values in a script

1. Make the results file active.
2. Choose **Results** > **Update Expected Value**.
3. Optionally, select **Run** > **Testcase** in order to run the test and confirm that it now passes. SilkTest replaces the expected values in the script with the actual values found at runtime.

Marking Failed Testcases

When a testplan results file shows testcase failures, you might choose to fix and then rerun them one at a time. You might also choose to rerun the failed testcases at a slower pace without debugging them to watch their execution more carefully.

Make the results file active and click **Results** > **Mark Failures in Plan**.

SilkTest marks all the failed testcases and makes the testplan the active file.

Merging results

You can merge results two different ways:

- Merging two results sets in a results file.
- Merging results of manual tests.

Navigating to errors

To find and expand the next error or warning message in the results file, choose **Edit** > **Find Error**. To skip warning messages and find error messages only, in the **Runtime Options** dialog, uncheck the check box labeled **Find Error stops at warnings**.

You can also use the **Find**, **Find Next**, and **Go to Line** commands on the **Edit** menu to navigate through a results file.

To expand an error message to reveal the cause of an error, click the red plus sign preceding the message. In addition to the cause, SilkTest displays the call stack, which is the list of 4Test functions executing at the time the error occurred.

SilkTest provides several ways to move from the results file to the actual error in the script:

- Double-click the margin next to an error line to go to the script file that contains the 4Test statement that failed.
- Click an error message and choose **Results** > **Goto Source**.
- Click an error message and press **Enter**.

To navigate from a testplan test description in a results file to the actual test in the testplan, click the test description and choose **Results** > **Goto Source**.

Viewing an individual summary

1. Click a testcase line in a suite or script results file, or click a test description in a testplan results file.
2. Choose **Results** > **Show Summary**.

Storing and Exporting Results

You can store and export results in a variety of ways:

- Store results in an unstructured ASCII format.
- Exporting results to a structured file for further manipulation.
- Sending the results directly to Issue Manager.

Storing results

SilkTest allows you to extract the information you want in an unstructured ASCII text format and send it to a printer, store it in a file, or look at it in an editor window.

To store results in an unstructured ASCII format

1. Choose **Results** > **Extract**.
2. In the **Extract To group** box on the **Extract Results** dialog, select the radio button for the destination of the extracted output: **Window (default)**, **File**, or **Printer**.
3. In the Include group box, check one or more check boxes indicating which optional text, if any, to extract. (This optional text is in addition to the output selected in the **Expand group** box.) The choices are:
4. Select a radio button in the **Expand** group box indicating which units to extract information about. Select **Scripts**, **Scripts and Testcases (default)**, or **Anything with Errors**.
5. Select one or more results sets from the **Results to Extract** group box.
6. Click **OK**.

   To learn how to save your results in a structured text file that an application such as a spreadsheet can process, see *Exporting structured information*. For information on exporting results to Issue Manager, see *Exporting Results to SilkCentral Issue Manager*.

Exporting Results to a Structured File for Further Manipulation

1. Click **Results** > **Export**. SilkTest displays the **Export Results** dialog.
2. Specify the file name. By default, SilkTest suggests the name `results file.rex` (for results export).
3. Specify which fields you want to export to the file.
4. Specify how you want the fields delimited in the file. The default is to comma delimit the fields and put quotations marks around strings.

   You can pick another built-in delimited style listed in the **Export format** list box or select **Custom** and specify your own delimiters.
5. To include header information in the file, check the **Write header** check box. Header information contains the name of the results file, which fields were exported, and how the fields were delimited.
6. To include the directory and file that stores the results file in the file, check the **Write paths relative to the results file** check box.
7. Specify which results sets you want to export. The default is the results set that is currently displayed in the results window.
8. Click **OK**. SilkTest saves the information in a delimited text file. You can import that file into an application that can process delimited files, such as a spreadsheet.

Removing the unused space in a results file

1. In SilkTest, open a results file.
2. Choose **Results** > **Compact**. SilkTest reduces the file's size.

Sending Results Directly to Issue Manager

SilkCentral Issue Manager is the defect-tracking product that you can use to create and manage bug reports, enhancement requests, and documentation issues for your application. Issue Manager is integrated with SilkTest. You can associate individual SilkTest tests with defects stored in Issue Manager and have Issue Manager process the defects based on the results of the tests.

You can pass your test results to Issue Manager in two ways:

- Sending results directly to Issue Manager - This is the easiest way to pass the results if you are running both Issue Manager and SilkTest.

- Exporting the results to a `.rex` file for importing later in Issue Manager.

`.rex` files can be read correctly by Issue Manager 3.2 (and above). While you can export a `.rex` file to previous versions of SilkRadar, syntax/data errors occur.

Logging Elapsed Time Thread and Machine Information

Using the **Runtime Options** dialog, you can specify that you want to log elapsed time, thread number, and current machine information. This information is then written to the results file where you can display and sort it. For example, if you encounter nested testcases in the results files because you use multithreading, check this check box to record thread number information in your results file. Then, you can sort the lines in your results file by thread number to better navigate within the nested testcases.

1. Choose **Options** > **Runtime** to open the **Runtime Options** dialog.
2. In the Results area, check the **Log elapsed time**, **thread**, and **machine for each output line** check box.
3. Click **OK**.

Presenting Results

Fully customize a chart

1. In SilkTest, generate the **Pass** > **Fail** report and click the **Chart** tab.
2. Click the area of the chart that you want to customize, for example, the text that appears for the title and footnote.
3. Double-click the selected area. SilkTest displays a dialog, showing the properties for the selected area. (You can also right-click anywhere on a chart and select the area you want to modify from a popup menu.)
4. Make your changes. For information about the properties, click **Help**.
5. Click **Apply** to apply the changes without closing the dialog. Click **OK** to close the dialog.

Generate a PassFail report on the active results file

1. Make sure the testplan results file you want to report on is active, and then choose **Results** > **Pass** > **Fail Report**.
2. On the **Results Pass/Fail Report** dialog select an attribute to report on from the **Subtotal by Attribute** drop-down list.
3. Click **Generate**. SilkTest displays the report in the list box.
4. Take one of the following actions:

Producing a PassFail chart

You can chart a generated **Pass/Fail** report, that is, produce report information as a graph, or you can directly graph the testplan results information without a preexisting report

1. Click the **Chart** tab. If you have already generated a report, SilkTest displays a chart of the generated report. (You might need to resize the window so there is enough room to display the chart well.) If you have not generated a report, SilkTest displays a default chart, which allows you to modify chart parameters before you actually generate the chart.

2. Take one of the following actions:

Change basic charting properties	1. Click **Setup**. The **Chart Settings** dialog is displayed. 2. To change the chart type, select an option from the **Chart Type** drop-down list box. SilkTest provides bar charts, line charts, and area charts. 3. Click **Apply** to update the chart and leave the **Chart Settings** dialog open. You can also choose whether the chart is three- dimensional, is stacked (for bar charts), and displays a legend, which describes the data being charted. SilkTest displays a model that represents how the chart will look based on current settings.
Add the results from another execution of the testplan to the chart	1. Click **Select**. The **Select Results** dialog is displayed, listing recent runs of the current testplan. SilkTest keeps a history of results for each testplan. The number of results it keeps is determined by the value for **History Size** in the **Runtime Options** dialog. 2. Select the results you want to add to the chart. The results from the selected execution of the testplan will be added to the results currently charted. You can use this feature to compare two different runs of the same tests to spot problem areas. You can chart today's results, then click **Setup** and select yesterday's results to have both appear on one chart.
Move a part of the chart	1. Click the part you want to move, such as the title, legend, or footnote (the text that displays below the chart). The area is selected. 2. Drag it with the mouse.
Print the chart	1. Click **Print**. The **Print Pass/Fail Chart** dialog displays. You can specify a header or footer. 2. Click **OK** to print the chart.
Copy the chart to the clipboard	1. Right-click anywhere on the displayed chart, and then click **Copy**. 2. The chart is placed on the clipboard. You can paste it into another application.
Change advanced charting properties	Usually you can get the chart you want using the default and basic charting properties. But if you want more customization, you can modify just about any property in the chart, including: • text that appears for the title and footnote • font used for any text in the chart • location for the title, legend, and footnote • colors used for the data • size and spacing of the bars in bar charts • borders and shading to the background (backdrop) of any area • See Customizing a chart.
Generate the chart	Once you are satisfied with the chart parameters, click **Generate**. The **Pass/Fail** chart is displayed.

Displaying a different set of results

1. Click **Results** > **Select**. SilkTest displays the **Select Results** dialog with the most current results set displayed first.
2. Select the set of results you want to see and click **OK**.

Debugging Test Scripts

Overview of the debugger

You will find out about many of the errors or inconsistencies in your scripts when SilkTest automatically raises an exception in response to them. Some problems, however, cause a script to work in unexpected ways, but do not generate exceptions. You can use the SilkTest debugger to solve these kinds of problems.

Using the debugger, you can step through a script a line at a time and stop at specified breakpoints, as well as examine local and global variables and enter expressions to evaluate.

But the SilkTest debugger is more than just a tool for fixing scripts. You can also use the debugger to help find problems in your application, using the debugging facilities to step through the application slowly so you can determine just where a problem occurs.

The Debugger allows you to view the results of your testing in the following ways:

- View the debugging transcript when you debug a script. See *Viewing the debugging transcript*. SilkTest records error information and output from the print statements in a transcript, not in a results file.
- Examine the debugging variables while you are debugging a test script. See *View variables*.
- View the call stack. The call stack is a description of all the function calls that are currently active in the script you are debugging. By viewing the call stack, you can trace the flow of execution, possibly uncovering errors that result when a script's flow of control is not what you intended. To view the current call stack, choose **View** > **Call Stack**. SilkTest displays the call stack in a new window. To return to the script being debugged, press F6 or choose **View** > **Module** and select the script from the list.

You cannot use the debugger from plan (`*.pln`) files, however, you could call testcases from a `main()` function and debug it from there.

You may not modify files when you are using the debugger. If you want to fix a problem in a file, you must first stop the debugger, and then make the fix.

Starting the debugger

There are several ways to enter the debugger:

Script in active window	Click **Run** > **Debug**. SilkTest enters the debugger and pauses. It does not set a breakpoint.
Another script	Click **File** > **Debug** and select the script file from the Debug dialog. SilkTest enters the debugger and pauses. It does not set a breakpoint.
A testcase	With a script active, click **Run** > **Testcase**, select a testcase from the **Run Testcase** dialog, and click **Debug**. SilkTest enters the debugger and sets a breakpoint at the first line of the testcase.
An application state	Click **Run** > **Apllication State**, select an application state from the **Run Application State** dialog, and click **Debug**.

	SilkTest enters the debugger and sets a breakpoint at the first line of the application state definition.
A plan file	You cannot use the debugger from plan files (`*.pln`) , however, you can call testcases from a `main()` function and debug it from there.

When you enter the debugger, you can execute the script under your control.

You cannot edit a script when you are in the debugger.

Overview of breakpoints

A breakpoint is a line in the script where execution stops, so that you can check the script's status. The debugger lets you stop execution on any line by setting breakpoints. A breakpoint is denoted as a large red bullet(·).

One useful way to debug a script is to pause it with breakpoints, observe its behavior and check its state, then restart it. This is useful when you are not sure what lines of code are causing a problem.

During debugging, you can:

- Set breakpoints on any executable line where you want to check the call stack.
- Examine the values in one or more variables.
- See what a script has done so far.

You cannot set breakpoints on blank lines or comment lines.

Adjust breakpoints

This topic contains the information about viewing and deleting breakpoints, as well as setting temporary breakpoints.

Setting breakpoints

You cannot set breakpoints on blank lines or comment lines.

You can set breakpoints in any of these ways.

The first line of a function (or testcase)	1. Click **Breakpoint** > **Add**. 2. Double-click a module name to have the functions declared in that module listed in the **Function** list box. 3. Double-click a function name to set a breakpoint on the first line of that function.
Any line in a function (or testcase)	Place the cursor on the line where you want to set a breakpoint and choose **Breakpoint** > **Toggle**. or Double-click in the left margin of the line.
A specific line in a script	1. Click **Breakpoint** > **Add**. 2. In the **Breakpoint** field, type the number of the line on which you want to set a breakpoint. (For example, entering 8 sets a breakpoint on the eighth line of the script.) 3. Click **OK**.

Setting temporary breakpoints

Click **Debug** > **Run To Cursor** to set a temporary breakpoint (indicated by a hollow red circle in the margin) on the line containing the cursor. SilkTest immediately runs the script, stopping at the current line. The breakpoint is cleared after it is hit.

Viewing a list of breakpoints

To view a list of all the breakpoints in a script, choose **View** > **Breakpoints**.

Deleting breakpoints

You can delete breakpoints in any of the following ways.

All breakpoints	1. Click **Breakpoint** > **Delete All**. 2. Click **Yes** to confirm the deletions.
An individual breakpoint	Place the cursor on the line where the breakpoint is set and choose **Breakpoint** > **Toggle** . or Double-click in the left margin of the line
One or more breakpoints	1. Click **Breakpoint** > **Delete**. 2. Select one or more breakpoints from the list box and click **OK**.

Overview of Expressions

If you type an identifier name, the result is the value that variable currently has in the running script. If you type a function name, the result is the value the function returns. Any function you specify must return a value, and must be in scope at the current line.

Properties and methods for a class are valid in expressions, as long as the declaration for the class they belong to is included in one of the modules used by the script being debugged.

If an expression evaluates to a complex value, like an array, SilkTest may display its result in collapsed form. Use **View** > **Expand Data** or **View** > **Collapse Data** (or double-click on the plus icon) to manipulate the display.

When a script reaches a breakpoint, you can evaluate expressions.

Evaluate expressions

1. Click **View** > **Expression**.
2. Type an expression into the input area and press **Enter** to view the result.

If you type an identifier name, the result is the value that variable currently has in the running script. If you type a function name, the result is the value the function returns. Any function you specify must return a value, and must be in scope at the current line.

Properties and methods for a class are valid in expressions, as long as the declaration for the class they belong to is included in one of the modules used by the script being debugged.

If an expression evaluates to a complex value, like an array, SilkTest may display its result in collapsed form. Use **View** > **Expand Data** or **View** > **Collapse Data** (or double-click the plus icon) to manipulate the display.

Debugger menus

In debugging mode, the menu bar includes three additional menus: **Debug**, **Breakpoint**, and **View**.

- Debug menu commands allow you to control the script's flow.
- Breakpoint menu commands add or remove a breakpoint.
- View menu commands display different elements of the running script (for example, local and global variables, the call stack, and breakpoints) and evaluate expressions.

Designing and testing with debugging in mind

Here are some suggestions for designing and testing a script that will facilitate debugging it later:

- Plan for debugging (and robustness) when you're designing the script, by having your functions check for valid input and output, and perform some operation that informs you if problems occur.
- Test each function as you write it, by building it into a small script that calls the function with test arguments and performs some operation that lets you know it works. Or use the debugger to step through the execution of each function individually after you have coded all (or part) of the script.
- Test each routine with the full range of valid data values, including the highest and lowest valid values. This is a good way to find errors in control loops.
- Test each routine with invalid values; it should reject them without crashing.
- Test each routine with null (empty) values. Depending on the purpose of the script, it might be useful if a reasonable default value were provided when input is incomplete.

Stepping into and over functions

Sometimes the key to locating a bug in your code is to divide the script up into discrete functions, and debug each function separately. One good way to do this is with the **Step Into**, **Step Over**, and **Finish Function** commands on the **Debug** menu. These commands let you run and test functions individually.

Step Into	Step through the function one line at a time, executing each line in turn as you go.
Step Over	Speed up debugging if you know a particular function is bug-free.
Finish Function	Execute the script until the current function returns. SilkTest sets the focus at the line where the function returns. Try using Finish Function in combination with Step Into to step into a function and then run it.

Viewing variables

To view a list of all the local variables that are in scope (accessible) from the current line, including their values, choose **View** > **Local Variables**.

To view a list of global variables, choose **View** > **Global Variables**. The variables and their values are listed in a new window.

If a variable is uninitialized, SilkTest labels it `<unset>`.

If a variable has a complex value, like an array, SilkTest might need to display its result in collapsed form. Use **View** > **Expand Data** and **View** > **Collapse Data** (or double-click the plus icon) to manipulate the display.

To return to the script being debugged, press F6 or choose View/Module and select the script from the displayed list.

Changing the value of variables

To change the value of an active variable, select the variable and type its new value in the **Set Value** field.

While viewing variables, you can also change their values to test various scenarios.

When you resume execution, SilkTest uses the new values.

Enabling view trace listing

You can have SilkTest record all the methods that a script or script invoked in the transcript. Each entry includes the method name and the arguments passed into the method. This information can be useful for debugging because it tells you exactly what GUI functions were actually called by the running script.

To correct this problem, change your tag in the frame file (or in the script if you are not using a frame file).

1. In SilkTest, click **Options** > **Runtime** to display the **Runtime Options** dialog.
2. Check the **Print Agent Calls** and the **Print Tags with Agent Calls** check boxes.
3. Run the script that produces the error. In addition to error information and output from print statements, the transcript also lists all methods called by the script.

   To check the Agent trace during debugging, when execution pauses, click **View** > **Transcript**.
4. Choose **Record** > **Window Identifier** and point to the window.
5. Note the tag or identifier.
6. Compare the tag/identifier with what is listed in the results file. What you should see is a mismatch of what is in the results file versus what you get in the **Record Window Identifier** dialog (which is how SilkTest is identifying the window).

   In the script:

   ```
   MainWin("Test Application").DialogBox("CheckBox").SetActive ()
   ```

   In the results file:

   ```
   MainWin("Test Application").DialogBox("CheckBox").SetActive () tag="/
   [MainWin]Test Application/[DialogBox]CheckBox"
   *** Error: Window '[DialogBox]CheckBox' was not found
   ```

   In the Record Window Identifier dialog:

   ```
   MainWin("Test Application").DialogBox("CheckBox")
   ```

   If you look closely, you can see that the **Record Window Identifier** dialog is telling us the actual tag for the DialogBox is Check Box (note the space between Check and Box). But if you look at the results file, the tag does not show the space. This is the problem.

Executing a script in the debugger

Once you have set one or more breakpoints, you can start executing your script.

1. Click **Debug** > **Run**. SilkTest runs the script until it hits the first breakpoint, an error occurs, or the script ends. SilkTest displays a blue triangle next to the line where it stopped running the script.
2. Click **Debug** > **Continue**. SilkTest runs the script until it hits the next breakpoint, an error occurs, or the script ends.
3. Click **Debug** > **Step Into**, **Debug** > **Step Over**, or **Debug** > **Finish Function** to run a smaller chunk of your script. See Stepping into and over functions for more information.

Viewing a list of modules

1. Click **View** > **Module**.
   SilkTest displays a list of modules in the **View Module** dialog. The list includes all the modules SilkTest loads at startup (that is, the modules loaded by `startup.inc`, including `winclass.inc`), so you can set breakpoints in GUI functions, window class declarations, and so forth.
2. Double-click a module's name to view it in a debug window.

View the debugging transcripts

Choose **View** > **Transcript** when execution is stopped.

SilkTest displays the transcript in a new window. To save its contents to a text file, choose **File** > **Save**.

The **Transcript** window has an **Execute** field that you can use to send commands to the application you are testing. You can type in any command that would be valid in a script and click **Execute**. For example, you might want to print the value of a variable or the contents of a window.

Working with scripts

To run the script you are debugging	Click **Debug** > **Run**. The script runs until a breakpoint is hit, an error occurs, or it terminates.
To reset a script	Click **Debug** > **Reset**. This frees memory, frees all variables, and clears the call stack. The focus will be at the first line of the script.
To stop execution of a running script	Press `Shift+Shift` when running a script on the same machine or choose **Debug** > **Abort** when running a script on a different machine.

Exiting the debugger

You can leave the debugger whenever execution is stopped.

To exit the debugger, choose **Debug** > **Exit**.

Debugging Tips

Checking the precedence of operators

The order in which 4Test applies operators when it evaluates an expression may not be what you expect. Use parentheses, or break an expression down into intermediate steps, to make sure it works as expected. You can use **View** > **Expression** to evaluate an expression and check the result.

Code that never executes

To check for code that never executes, step through the script with **Debug** > **Step Into**. See the Debug menu for more information.

Global and local variables with the same name

It is usually not good programming practice to give different variables the same names. If a global and local variable with the same name are in scope (accessible) at the same time, your code can access only the local variable.

To check for repeated names, use **View** > **Local Variables** and **View** > **Global Variables** to see if two variables with the same name are in scope simultaneously.

Global variables with unexpected values

When you write a function that uses global variables, make sure that each variable has an appropriate value when the function exits. If another function uses the same variable later, and it has an unexpected value on entry to the function, an error could occur.

To check that a variable has a reasonable value on entry to a function, set a breakpoint on the line that calls the function and use the command **View** > **Global Variables** to check the variable's value.

Incorrect use of break statements

A `break` statement transfers control of the script out of the innermost nested `for`, `for each`, `while`, `switch`, or `select` statement only. `Break` exits from a single loop level, not from multiple levels. Use **Debug** > **Step Into** to step through the script one line at a time and ensure that the flow of control works as you expect. See **Debug** menu for more details.

Incorrect values for loop variables

When you write a `for` loop or a `while` loop, be sure that the initial, final, and step values for the variable that controls the loop are correct. Incrementing a loop variable one time more or less than you really want is a common source of errors.

To make sure a control loop works as you expect, use **Debug** > **Step** Into to step through the execution of the loop one statement at a time, and watch how the value of the loop variable changes using **View** > **Local Variables**. See **Debug** menu for more details.

Infinite loops

To check for infinite loops, step through the script with **Debug** > **Step Into**. See **Debug** menu for more details.

Typographical errors

It is easy to make typographical errors that the 4Test compiler cannot catch. If a line of code does nothing, this might be the problem.

Uninitialized variables

SilkTest does not initialize variables for you. So if you have not initialized a variable on entry to a function, it will have the value `<unset>`. It is better to explicitly give a value to a variable than to trust that another function has already initialized it for you. Also, remember that 4Test does not keep local variables around after a function exits. The next time the function is called, its local variables could be uninitialized.

If you are in doubt about whether a variable has a reasonable value at a particular point, set a breakpoint there and use **View** > **Global Variables** or **ViewLocal Variables** to check the variable's value.

Troubleshooting

ActiveX-VB Applications

What happens when you enable ActiveXVisual Basic

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest updates to use the appropriate Visual Basic/ActiveX include file and merge the Visual Basic/ ActiveX property sets and Help text for the **Library Browser**.

Troubleshooting tips for ActiveXVisual Basic controls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Check property sets initialization file

If when you try to verify properties, SilkTest does not display the appropriate Visual Basic properties, check your property set initialization file, `vbprpset.ini`, in the SilkTest installation directory. When you open the file in an editor, you should see familiar Visual Basic properties. If you don't, reinstall SilkTest with enhanced Visual Basic and ActiveX support.

SilkTest does not recognize ActiveX controls in a Web application

If you notice that SilkTest is not recognizing the ActiveX controls in your Web application when recording or playing back tests, you should make sure that the ActiveX/Visual Basic support is enabled for your browser. See Enabling extensions.

SilkTest displays an error when playing back a click on a Sheridan command button (OLESSCommand)

Problem: In SilkTest, if you record a click against a Sheridan command button (OLESSCommand), SilkTest does not play back the click and records an error in the results file for the test.

Cause: The OLESSCommand should be a windowless control.

Action to take: In the `VBclass.inc` file, edit the class declaration for OLESSCommand to have it inherit from `WindowlessControl` instead of `Pushbutton`.

Objects showing as CustomWins

There are two reasons why SilkTest may not record declarations for native Visual Basic objects, but instead records them as `CustomWins` (custom windows):

- You didn't follow the procedure for recording classes. See *Recording new classes*.
- The extension is not being loaded property into the application. See *Enabling the ActiveX/Visual Basic extension*.

Record Class finds no properties or methods for an object

If Record Class finds no properties or methods for an object:

- Make sure the extension is load and enabled properly. See Enabling extensions and Enabling the ActiveX/Visual Basic extension.
- Verify that the object whose class you are recording is an Active control or a native Visual Basic control.
- If the ActiveX control was created in Visual Basic 5, it must expose its properties and methods. Refer to your Visual Basic 5 documentation for more information.

Inconsistent recognition of ActiveX controls

In certain cases, SilkTest may see ActiveX controls (which often have a native class beginning with OLE) as having the native class ATL:<variable hex #>, or may see the ActiveX controls as children of an ATL:<variable hex #> control. In other cases, the parent control of an ActiveX control may seem to disappear on occasion. In these situations, there are a couple of settings in extend\axext.ini that may help.

- If the inconsistent recognition problem involves ATL controls, then first try setting DontIgnore=ATL in the [VBOptions] section.
- If the recognition problem occurs when SilkTest invokes the application under test, especially if it seems slow to render completely, then try to kill the SilkTest Agent while leaving the application running, and then restart the Agent. If SilkTest recognizes the control properly, then you should be able to correct the problem without having to kill the Agent by setting `AxextDelay=<n>` in the `[VBOptions]` section, where <n> is the number of seconds that the application should require to start up completely.

  Note that the amount of time that SilkTest requires to recognize the application when it is invoked will be extended by `AxextDelay` seconds, so you should not make the number too large. In addition, if the application contains frame classes such as `AfxFrameOrView42` or `AfxMDIFrame42`, then you should try class-mapping them to Ignore. Ignoring those windows may eliminate some unnecessary layers and also make SilkTest's recognition of the window hierarchy more consistent.

Troubleshooting Test Failures for VB Application Configuration

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

The test of your VB application may have failed for one of the reasons below. If the following suggestions do not address the problem you are having, you can enable your Extension manually. See *Enabling extensions*.

The application may not have been ready to test

1. Click **Enable Extensions** on the **Basic workflow** bar.
2. On the **Enable Extensions** dialog box, select the application for which you want to enable extensions.
3. Close and restart your application. Make sure the application has finished loading, and then click **Test**.

You may have another configured VB application open

You must close all configured VB applications before you can configure another VB application.

ApplicationEnvironment

Dr. Watson when Running from Batch File

This functionality is available only for projects or scripts that use the Classic Agent.

If you get a Dr. Watson error when trying to record window declarations for a Java application launched through a batch file, the classpath set in the batch file is overriding the global classpath. Make sure the classpath in the batch file points to the appropriate SilkTest .jar file.

I Cannot get SilkTest to Work with JBuilder or Oracle JDeveloper

This functionality is available only for projects or scripts that use the Classic Agent.

Make sure that you have configured SilkTest properly for these environments.

SilkTest does not Launch my Java Web Start Application

This functionality is available only for projects or scripts that use the Classic Agent.

Sometimes the default base state does not launch your Java Web Start application. If your application requires Java Web Start to launch, then you must manually edit the constant *sCmdLine* in your declaration for the main application window, which is designated as the *wMainWindow*.

Create New Test Frame detects the Windows command line that directly invoked the main window, so it will set *sCmdLine* to the JRE command line launched by Java Web Start. However, since you want to start Java Web Start instead of directly starting the JRE, you must edit *sCmdLine* to launch the Java Web Start executable, `javaws.exe`, with the .jnlp file for your application.

Example

```
const sCmdLine = "C:\Program Files\Java Web Start\javaws.exe C:
\MyAppDir\MyJWSApp.jnlp"
```

Which JAR File do I Use

This functionality is available only for projects or scripts that use the Classic Agent.

For JDK/JRE 1.3, 1.4, and higher, use **SilkTest_Java3.jar**.

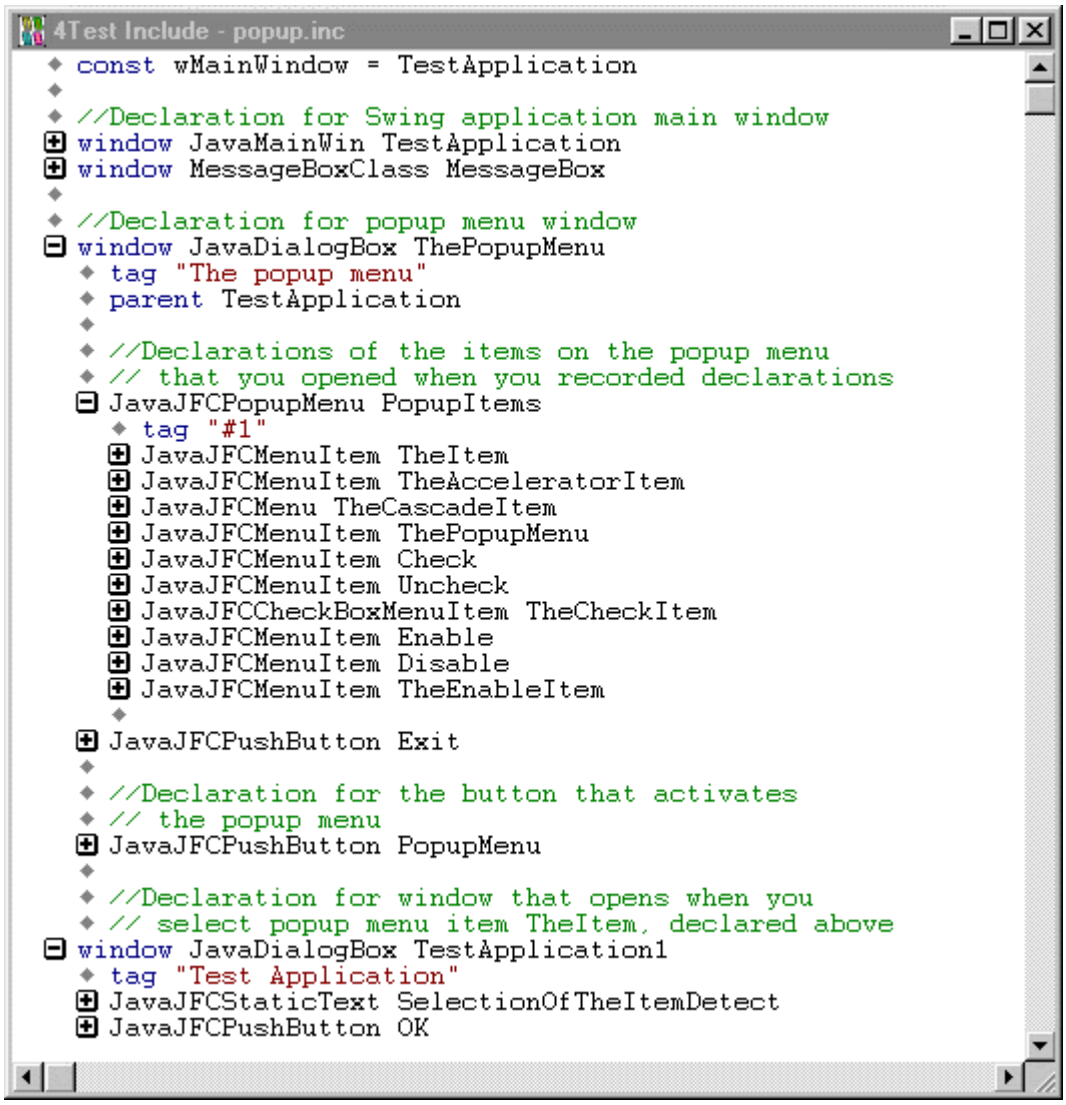
For additional information about supported Java versions, refer to the *Release Notes*. You can access the *Release Notes* by clicking **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes**.

Sample Declarations and Script for Testing JFC Popup Menus

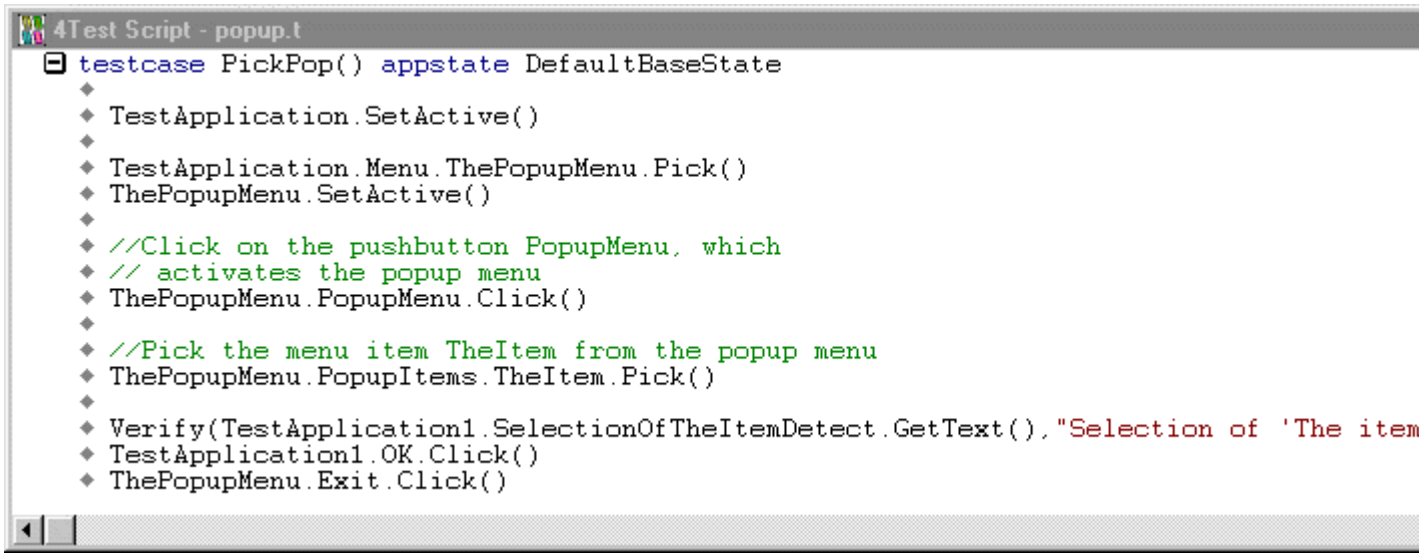
This functionality is available only for projects or scripts that use the Classic Agent.

When you record `Pick`() for a Java popup menu item, make sure that the recorder window shows the proper menu item. If the recorder seems to stick on the wrong menu item, move the mouse off of the popup menu and then back on to it. That should force the recorder to update.

Sample test frame:



Sample test script:



Java Extension Loses Injection when Using Virtual Network Computing (VNC)

This functionality is available only for projects or scripts that use the Classic Agent.

The Java extension can periodically stop recognizing Java objects on JFC (Swing) applications if the Agent machine, where the Java application is running, is being "viewed" remotely through Virtual Network Computing (VNC), a popular tool that lets you view and control a remote machine.

Symptoms of this problem include the following:

- The main window is viewed as a `JavaDialogBox`.
- An `*** Error: Window '[JavaMainWin]<window name>' was not found` message box displays.
- `JavaJFCMenu` items are not seen, resulting in an error such as `*** Error: Window JavaJFCMenu <menu name>' was not found`.

Avoid using VNC to view the automation which is running.

Browsers

I am not Testing Applets but Browser is Launched During Playback

This functionality is available only for projects or scripts that use the Classic Agent.

When you enable a browser extension in SilkTest, the recovery system automatically launches the enabled browser when returning to DefaultBaseState. During sessions when you are testing only standalone Java applications, we recommend that you set **Default browser** to `none` in the **Runtime Options** dialog box.

Playback is Slow when I Test Applications Launched from a Browser

This functionality is available only for projects or scripts that use the Classic Agent.

When you test applications, not applets, launched from a browser, the recovery system performs several functions that might impact performance, including the following:

- Closes the Java application when exiting each test case.
- Loads the URL of the launch site and relaunches the Java application when entering each test case.

To avoid opening and closing the Java application for each test case, you can add the following code to your test frame:

1. Add an `Invoke`() method inside the `JavaMainWin` definition that will launch the Java application.
2. Inside the *wMainWindow* definition of your test frame, assign the constant *lwLeaveOpen* to the main window of the Java application.
3. Add `TestCaseEnter`() and `TestCaseExit`() methods at the top of the test frame.

Library Browser does not display web browser classes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

The Library Browser does not display any of the classes for web browsers.

This problem can be caused by either of two conditions:

- The extensions for the web browser you are testing have not been enabled.
- SilkTest was not able to successfully compile the files that populate the Library Browser with the browser classes.
- Successful compilation incorporates the information in the SilkTest browser include file (for example, the `extend\netscape.inc` file). The browser include file pulls in the appropriate help file (`.ht`) to populate the Library Browser with the additional classes (`extend\netscape.ht`).

Solution

1. Make sure that the browser extensions are enabled on the target and host computers.
   - On the target computer, from the SilkTest program group, choose **Extension Enabler**. Check the **Primary Extension** for the browser to make sure it says `Enabled`.
   - On the host computer, from SilkTest, choose **Options** > **Extensions**. Check the **Primary Extension** for the browser to make sure it says `Enabled`.
2. Check the SilkTest **Runtime Options** dialog. (In SilkTest, choose **Options** > **Runtime**.) Make sure the **Use Files** field contains the browser include file (for example, `netscape.inc`).
3. Exit SilkTest and restart it.
4. This action causes SilkTest to recompile the include files.

Error Messages

Agent not responding

Problem

You get the following error message:

```
Error: Agent not responding
```

This error can occur for a number of reasons.

Solution

Try any or all of the following:

- Restart the application that you are testing
- Restart SilkTest
- Restart the Host machine

If you are recording declarations on a very large page and get this error, consider increasing the AgentTimeout.

BrowserChild MainWindow Not Found When Using Internet Explorer 7.x

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

My scripts are failing in Internet Explorer 7.x with the following error `[BrowserChild] MainWindow not found`.

Solution

When recording a new frame file using Set Recovery System, by default SilkTest does not explicitly state that the parent of the window is a browser. To resolve this issue, the "parent Browser" line must be added to the frame file. For example:

```
[-] window BrowserChild Google
[ ] tag "Google"
[ ] parent Browser
```

Cannot find file agent.exe

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

Running a script gives the following error:

```
Cannot find file agent.exe [or one of its components]. Check to ensure the
path and filename are correct and that all required libraries are available.
```

Solution

1. Exit SilkTest (this automatically shuts down the Agent).
2. Explicitly start the Agent by itself.
3. Restart SilkTest.

Control is not responding

Problem

You run a script and get the following error: `Error: Control is not responding`

This is a catch-all error message. It usually occurs in a `Select( )` statement when SilkTest is trying to select an item from a ListBox, TreeView, ListView, or similar control.

The error can occur after the actual selection has occurred, or it can occur without the selection being completed. In general the error means that the object is not responding to the messages SilkTest is sending in the manner in which it expects.

Solution

Try these things to eliminate the error message:

- If the line of code is inside a `Recording` block, remove the `Recording` keyword.
- Set the following option just before the line causing the error: `Agent.SetOption(OPT_VERIFY_RESPONDING, FALSE).`
- If the selection is successful, but you still get the error, try using the `Do . . . except` feature.

Unable to Connect to Agent

Problem

You get the following error message: `Error: Unable to connect to agent`

This error can occur for a number of reasons.

Solution

Connect to the default agent	In SilkTest, choose **Tools** > **Connect to Default Agent**. The command starts the Classic Agent or the Open Agent on the local machine depending on which agent is specified as the default in the **Runtime Options** dialog. If the Agent does not start within 30 seconds, a message is displayed. If the default Agent is configured to run on a remote machine, you must connect to it manually.
Restart the agent that you require for testing	Choose **Start** > **Programs** > **Silk** > **SilkTest <version>** > **SilkTest Classic Agent** or **SilkTest Open Agent**.

Unable to Delete File dialog box

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

While you are using the Basic Workflow to configure a Java application, SilkTest is unable to delete the JVM files

You may have another application running that uses the same JVM as the application being configured or an application may be slow to release the JVM.

Close all applications that use the JVM (if any) and click **Retry**. That should give the application enough time to release the JVM.

If the suggestion above does not solve the problem, you can enable your extension manually.

Unable to Start Internet Explorer

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

While trying to record or run a testcase you get the following message: `Error: Unable to start Internet Explorer #`

This error occurs because the extensions enabled in the **Extensions Enabler** and **Options** > **Extensions** do not match the default browser.

Solution

Review the settings for your default browser and make them consistent with the settings for the host machine in **Options** > **Extensions** and for the target machine in the Extension Enabler. See Verfiying extension settings and Enabling extensions.

Variable Browser not defined

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

If you are encountering the following message: `Error: Variable Browser is not defined`.

This error occurs because no browser extensions have been enabled.

Solution

Enable at least one browser extension.

Window Browser does not define a tag

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

While trying to run or record testcases you get the following message: `Error: Window Browser does not define a tag for <Operating System>`.

This error occurs because no default browser has been specified.

Solution

You need to set a default browser. Whenever you record and run testcases, you need to have a default browser set so SilkTest knows which browser to use. If you didn't choose a default browser during SilkTest installation or if want to change the default browser please see Specifying your default browser.

Window is not active

Problem

You run a script and get the following error: `Error: Window 'name' is not active`.

This error means that the object SilkTest is trying to act on is not active. This message applies to top-level windows (`MainWin`, `DialogBox`, `ChildWin`).

Solution

You can correct the error by doing one of the following:

1. Edit the script and add an explicit `SetActive()` statement to the window you are trying to act on just above the line where the error is occurring. An easy way to do this is to double-click the error in the results file. You will be brought to the line in the script. Insert a new line above it and add a line ending with the `SetActive()` method.
2. Tell SilkTest not to verify that windows are active. There are two ways to do this:

   To turn off the verification globally, uncheck the **Verify that windows are active** option on the **Verification** tab in the **Agent Options** dialog (**Options** > **Agent**).

   To turn off the option in your script on a case by case basis, add the following statement to the script, just before the line causing the error:

   ```
   Agent.SetOption(OPT_VERIFY_EXPOSED, FALSE).
   ```
3. Then add the following line just after the line causing the error:

   ```
   Agent.SetOption(OPT_VERIFY_EXPOSED, TRUE)
   ```

   This means SilkTest will execute the action regardless of whether the window is active.
4. Extend the window time out to be greater than 10 by inserting the **Agent - Window Timeout** to >= 10 into your `partner.ini`.

Window is not enabled

Problem

You run a script and get the following error: `Error: Window 'name' is not enabled`.

This error means that the object that SilkTest is trying to act on is not enabled. This message applies to controls inside top-level windows (such as PushButton and CheckBox).

Solution

You can correct this problem in one of two ways.

- If the object is indeed disabled, edit the script and add the actions that will enable the object.
- If the object is in fact enabled and you want the script to perform the action, tell SilkTest not to verify that a window is enabled:

To turn off the verification globally, uncheck the **Verify that windows are enabled** option on the Verification tab in the **Agent Options** dialog (**Options** > **Agent**).

To turn off the option in your script on a case-by-case basis, add the following statement to the script, just before the line causing the error: `Agent.SetOption(OPT_VERIFY_ENABLED, FALSE)`

Then add the following line just after the line causing the error:
`Agent.SetOption(OPT_VERIFY_ENABLED, TRUE)`.

This means SilkTest will execute the action regardless of whether the window is enabled.

Window is not exposed

Problem

You run a script and get the following error: `Error: Window 'name' is not exposed`.

Sometimes, applications are written such that windows are hidden to the operating system, even though they are fully exposed to the user. A running script might generate an error such as `Window not exposed`, even though you can see the window as the script runs.

Solution

While it might be tempting to simply turn off the checks for these verifications from the **Agent Options/ Verification** dialog, the best course of action is to take such errors on a case by case basis, and only turn off the verification in cases where the window is genuinely viewable, but SilkTest is getting information from the operating system saying the object is not visible.

1. Add the following statement to the script, just before the line causing the error:
   `Agent.SetOption(OPT_VERIFY_EXPOSED, FALSE).`
2. Then add the following line just after the line causing the error:
   `Agent.SetOption(OPT_VERIFY_EXPOSED, TRUE).`

This means SilkTest will execute the action regardless of whether it thinks the window is exposed.

Window not found

Problem

You run a script and get the following error: `Error: Window 'name' was not found`.

Resolution

This error occurs in the following situations:

When the window that SilkTest is trying to perform the action on is not on the desktop.	If you are watching the script run, and at the time the error occurs you can see the window on the screen, it usually means the tag that was generated is not a correct tag. This could happen if the application changed from the time the script or include file was originally created. To resolve this issue, enable view trace listing in your script.
The window is taking more than the number of seconds specified for the window timeout to open.	To resolve this issue, set the **Window Timeout** value to prevent Window Not Found exceptions
In the TrueLog Options dialog, if all of the following options are set	• The action `PressKeys` is enabled. • Bitmaps are captured after or before and after the `PressKeys` action. • `PressKeys` actions are logged. The preceding settings are set by default if you select Full as the TrueLog preset. To resolve this issue, modify your testcase.

Functions and Methods

Class not Loaded Error

This functionality is available only for projects or scripts that use the Classic Agent.

You should not simply add the .jar file containing the Java class referenced in the `InvokeJava`() call to the directory containing the .jar files used by your application. `InvokeJava`() will be able to load your Java class only if it can be loaded by one of the following class loaders. If you receive a `Class Not Loaded` error when calling `InvokeJava`():

1. By default, InvokeJava() uses the ext class loader. To use this loader, the .jar file containing the Java class referenced in the `InvokeJava()` call should reside in the JVM's `lib\ext` directory with SilkTest's .jar file, which is `SilkTest_Java3.jar`.

   The class should be in the root directory of the .jar file, so that there is no path information in the .jar file.
2. Alternatively, `InvokeJava()` can use the application class loader. To use this loader, the .jar file containing the Java class referenced in the `InvokeJava()` call should be part of the application's classpath.

Exists Method Returns False when Object Exists

This functionality is available only for projects or scripts that use the Classic Agent.

There is a timing issue in Java that causes a delay when Java applications or applets render objects. To ensure that the `Exists` method detects Java objects, call it with a timeout parameter. You might need to experiment with different timeout values to find the one that works best for your system configuration, and application or applet.

For example `Verify (JavaAwtPushButton.Exists(1), TRUE)`.

How to Define TestCaseEnter and TestCaseExit Methods

This functionality is available only for projects or scripts that use the Classic Agent.

Add one `TestCaseEnter` and one `TestCaseExit` method at the top of your test frame to ensure that the Java application is not opened and closed unnecessarily. `TestCaseEnter` calls the Invoke method you defined earlier to launch the Java application only if it is not already running.

Note: `DefaultTestCaseEnter` and `DefaultTestCaseExit` are also called to retain the benefits of the recovery system in resetting the application to a base state.

Example

The following code sample shows how the `TestCaseEnter` and `TestCaseExit` methods are inserted at the top of a test frame:

```
TestCaseEnter()
DefaultTestCaseEnter()
if ( ! TestApp.Exists() )
  TestApp.Invoke()
TestApp.SetActive()
TestCaseExit(BOOLEAN bTRUE)
  Browser.SetActive()
DefaultTestCaseExit(bTRUE)

const wMainWindow = JavaAWTTestApplet
  window BrowserChild JavaAWTTestApplet
  window JavaMainWin TestApp
…
```

How to Define lwLeaveOpen

This functionality is available only for projects or scripts that use the Classic Agent.

Add the constant *lwLeaveOpen* inside the `wMainWindow` declaration to instruct the recovery system to leave the Java application open when returning to base state. You must assign this constant to the identifier of the main window of the Java application.

Example

The declaration for the main window of the Java application looks like the following:

```
window JavaMainWin TestApp
```

Following is the constant *lwLeaveOpen* inserted in the header section of the `wMainWindow` declaration.

Note: The constant *lwLeaveOpen* is assigned to *TestApp*, the identifier of the main window of the Java application.

```
window BrowserChild JavaAWTTestApplet
  tag "Java 1.1 AWT TestApplet"

  // The URL of this page
  const sLocation = "…"

  // The login user name
  // const sUserName = ?

  // The login password
  // const sPassword = ?

  // The size of the browser window
  // const POINT BrowserSize = {600, 400}

  // Sets the browser font settings to the default
  // const bDefaultFont = TRUE

  const lwLeaveOpen = {TestApp}
```

How to Write the Invoke Method

This functionality is available only for projects or scripts that use the Classic Agent.

Add an invoke method inside the `JavaMainWin` definition. This method should interact with the appropriate controls on the HTML page to launch the Java application from the browser. You can code this method by hand or click **Record** > **Method** to use the dialog box.

Example

An HTML page uses a pushbutton inside an applet to launch a standalone Java application from the browser.

The declaration for the `wMainWindow` looks like the following:

```
window BrowserChild JavaAWTTestApplet
```

The declaration for the pushbutton inside the applet looks like the following:

```
JavaApplet StartTheTestApplicationIn2
  tag "Start the Test Application in a"
JavaAwtPushButton StartTestApplication
tag
"Start Test Application"
```

The following is a sample `Invoke` method highlighted in blue and declared inside the declaration for the main window of the Java application.

Note: The method clicks on the pushbutton that launches the Java application.

```
window JavaMainWin TestApp
  tag "Test Application"
    Menu File
    Menu Control
    Menu Menu
    Menu DisabledMenu

    void OpenWindows (STRING sMenuItem)
    void Invoke()

JavaAWTTestApplet.StartTheTestApplicationIn2.StartTestApplicatio
n.Click()
```

I cannot Verify $Name Property during Playback

This functionality is available only for projects or scripts that use the Classic Agent.

If you do not explicitly set the native `$Name` property of a Java control, Java (AWT) assigns a different default name to each instance of the control. As a result, you will not be able to verify the `$Name` property of the control because it will have a different name each time you run your test, and each time you close and reopen it during one test run.

When you want to verify the `$Name` property of a Java control, explicitly name the control by adding a call to the native method `setName` in your script before each call to `VerifyProperties`.

If you do not want to verify the name of a Java control, make sure you uncheck the `$Name` property in the **Verify Window** dialog box.

Example

VerifyProperties will fail in the following script because the `JavaAwtCheckbox` *TheCheckBox* is assigned a different `$Name` during playback than when the test case was recorded:

```
testcase Test1 () appstate none
  recording
    TestApplication.SetActive ()
    TestApplication.Control.CheckBox.Pick ()
    xCheckBox.TheCheckBox.VerifyProperties ({...})
      ""
      {...}
        {"$Name", "checkbox0"}
    xCheckBox.SetActive ()
    xCheckBox.Exit.Click ()
```

We can make the test run successfully by explicitly setting the name of *TheCheckBox* before verifying properties, as shown in the following script:

```
testcase Test1 () appstate none
  recording
    TestApplication.SetActive ()
    TestApplication.Control.CheckBox.Pick ()
    xCheckBox.TheCheckBox.setName("checkbox0")
    xCheckBox.TheCheckBox.VerifyProperties ({...})
      ""
      {...}
        {"$Name", "checkbox0"}
```

```
    xCheckBox.SetActive ()
    xCheckBox.Exit.Click ()
```

I Get Errors when I Call Nested Methods

This functionality is available only for projects or scripts that use the Classic Agent.

You won't be able to call nested methods when any of the intermediate return objects are not 4Test-compatible. To work around this problem, add the method `invokeMethods` by hand to your test script. This method allows you to call nested methods inside Java.

Methods Return Incorrect Indexed Values in My Scripts

This functionality is available only for projects or scripts that use the Classic Agent.

There is an incompatibility in indexing between 4Test methods and native Java methods for classes such as `ListBox`, `PopupList`, and `ScrollBar`. 4Test methods are 1-based; Java native methods are 0-based. If you mix 4Test methods and native methods in a test script where you retrieve indexed values, you must compensate for the difference in indexing schemes to maintain the integrity of your test results.

We recommend that you do not mix methods in test scripts if at all possible. A good rule is that when both types of methods are available for all controls in your applications, use the 4Test methods only.

In situations where 4Test methods are not available for some of your classes and you must mix in native methods, use the following precautions when writing code to get indexed values:

- Do not pass an index of 0 to a 4Test method.
- Adjust indexes accordingly.

For `ListBox` *Lb*, the native Java AWT list method call *Lb*.`getItem`(*n*) retrieves the same value as the 4Test list method call *Lb*.`GetItemText`(*n*+1), but not the same value as *Lb*.`GetItemText`(*n*).

How can I Determine the Exact Class of a java.lang.Object Returned by a Method

This functionality is available only for projects or scripts that use the Classic Agent.

Many Java methods return values of type `java.lang.Object`. In order to call such methods in SilkTest, you must use `invokeMethods`() to call a method on the return object. Eventually, you must call a method that returns a 4Test-compatible value. However, you need to know the exact class of the `java.lang.Object` in order to know which methods are available for that object. Otherwise, you can only call `java.lang.Object` methods, which is a fairly limited list.

If your method returns a `java.lang.Object` value, you can use the following `invokeMethods`() call to return the name of the class of the return object:

```
STRING sClass = wObj.invokeMethods({"<method that returns java.lang.Object>",
  "getClass", "getName"}, {{<parameter list for the method of interest>}, {},
{}})
```

The above statement will call the following 3 Java methods:

Method	Description
<method that returns java.lang.Object> (<parameter list for the method of interest>)	This is the method of interest. It returns a value of type `java.lang.Object`, which is not 4Test-compatible and therefore must be used to call a new method. **Record Class** only lists this method if you check **Show all**

Method	Description
	methods. The method displays in the commented list below the declaration of `wObj.invokeMethods()`.
getClass()	This `java.lang.Object` method returns a value of type `java.lang.Class`, which is not 4Test-compatible and therefore must be used to call a new method.
getName()	This `java.lang.Object` method returns a value of type `java.lang.String`, which is 4Test-compatible.

Once you know the name of the class, you can call methods specific to that class. Preferably those methods will return a 4Test-compatible type. Otherwise, you will need to chain additional methods in the `wObj.invokeMethods()` call.

Expanding on the previous example:

```
STRING sClass = wObj.invokeMethods({"<method that returns java.lang.Object>",
  "getClass", "getName"}, {{<parameter list for the method of interest>}, {},
{}})
  ANYTYPE aProp1
  ANYTYPE aProp2
  switch sClass
    case "ClassA"
      aProp1 = wObj.invokeMethods({"<method that returns java.lang.Object>",
        "getClassAProperty1"}, {{<parameter list for the method of
interest>}, {}})
      aProp2 = wObj.invokeMethods({"<method that returns java.lang.Object>",
        "getClassAProperty2"}, {{<parameter list for the method of
interest>}, {}})
    case "ClassB"
      aProp1 = wObj.invokeMethods({"<method that returns java.lang.Object>",
        "getClassBProperty1"}, {{<parameter list for the method of
interest>}, {}})
      aProp2=wObj.invokeMethods({"<method that returns java.lang.Object>",
        "getClassBProperty2"}, {{<parameter list for the method of
interest>}, {}})
    default
      RaiseError (E_UNSUPPORTED, "java.lang.Object is of an unknown class:
{sClass}")
```

`toString()` is a useful general method. It is a `java.lang.Object` method that returns a value of type `java.lang.String` and which translates to 4Test type STRING. You may be able to use `toString()` to return a value when you do not really understand what the previous method does. However, `toString()` may return a blank string, or the value may not make sense.

Multiple Machines Testing

Remote testing and default browser

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest uses the registry to determine which version of IE is installed on your machine; this means that the correct default browser is selected when you choose Internet Explorer 6 or Internet Explorer 7.

If you are doing remote testing, you will have to set up your default browser for your target test machine yourself. The reason this is because both versions of Internet Explorer use the same `domex.dll` and therefore SilkTest does not select the default browser at all.

Setting up the recovery system for multiple local applications

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

Problem

By default, the recovery system will only work for the single application assigned to the const `wMainWindow`. With distributed testing, you can get recovery on multiple applications by using `multitestcase` instead of `testcase`.

You might ask whether you can get the recovery system to work on multiple applications that are running locally using `multitestcase` locally. The answer is no; `multitestcase` is for distributed testing only.

But you can use the following solution instead, using `testcase`.

Solution

To get recovery for multiple local applications, set up your frame file to do the following:

1. Get standard `wMainWindow` declarations for each application. The easiest way is to select **File** > **New** > **Test Frame** for each application, then combine the `wMainWindow` declarations into a single frame file or include them with use.
2. Make the global `wMainWindow` a variable of type `WINDOW`, rather than a constant.
3. Assign one of the windows to `wMainWindow` as a starting point.
4. Create a LIST OF WINDOW and assign the `wMainWindow` identifier for each application you are dealing with to it.
5. Define a `TestcaseEnter` function so that you reassign the `wMainWindow` variable and call `SetAppState` on each `MainWin` in turn.
6. Define a `TestcaseExit` function so that you reassign the `wMainWindow` variable and call `SetBaseState` on each `MainWin` in turn.
7. Then use `DefaultBaseState`, or your own base state if you want, with each of your testcases. In your testcase, use `SetActive` each time you switch from one application to the other.

The example shows how this would work in the case of two demo applications shipped with SilkTest (Text Editor and Test Application). To see that the recovery system is working for both applications, turn on the two debugging options in **Runtime Options** and look at the transcript after running the test script.

The first testcase has an intentional error in its last statement to demonstrate the recovery system. The testcase also demonstrates how to move data from one application to another with `Clipboard.GetText` and `Clipboard.SetText`.

Because the recovery system is on, the DefaultBaseState will take care of invoking each application if it is not already running and will return to the DefaultBaseState after each testcase, even if the testcase fails.

The example consists of two files, `two_apps.t` and `two_apps.inc`. You can print them out or copy them to the **Clipboard**, then paste them into SilkTest (you might have to do some cleanup where the indentation of lines is incorrect in the pasted file).

two_apps.t

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

```
[-] testcase Test1 () appstate DefaultBaseState

 [ ] //SetActive each time you switch apps

 [ ] TestApplication.SetActive()
```

```
 [ ] TestApplication.File.New.Pick ()

 [ ] MDIChildWindow1.TextField1.SetPosition (1, 1)

 [ ] MDIChildWindow1.TextField1.TypeKeys ("In Test Application MDI Child
Window #1.")

 [ ] //SetActive each time you switch apps

 [ ] TextEditor.SetActive ()

 [ ] TextEditor.File.New.Pick ()

 [ ] TextEditor.ChildWin("(untitled)[1]").TextField("#1").TypeKeys ("In Text
Editor untitled Document window.<Enter>")

 [ ] //SetActive each time you switch apps

 [ ] TestApplication.SetActive()

 [ ] LIST OF STRING lsTempStrings

 [ ] lsTempStrings = MDIChildWindow1.TextField1.GetMultiText()

 [ ] Clipboard.SetText([LIST OF STRING]lsTempStrings)

 [ ] //SetActive each time you switch apps

 [ ] TextEditor.SetActive()

 [ ] TextEditor.ChildWin("(untitled)
[1]").TextField("#1").SetMultiText(Clipboard.GetText(),2)

 [ ] TextEditor.VerifyCaption("FooBar")

[-] testcase Test2 () appstate DefaultBaseState

 [ ] wMainWindow = TestApplication

 [ ] TestApplication.SetActive()

 [ ] TestApplication.File.New.Pick ()

 [ ] MDIChildWindow1.TextField1.SetPosition (1, 1)

 [ ] MDIChildWindow1.TextField1.TypeKeys ("In Test Application MDI Child
Window #1.")

 [ ] wMainWindow = TextEditor

 [ ] TextEditor.SetActive ()

 [ ] TextEditor.File.New.Pick ()

 [ ] TextEditor.ChildWin("(untitled)[1]").TextField("#1").TypeKeys ("In Text
Editor untitled Document window.<Enter>")

 [ ] wMainWindow = TestApplication

 [ ] TestApplication.SetActive()

 [ ] LIST OF STRING lsTempStrings
```

```
 [ ] lsTempStrings = MDIChildWindow1.TextField1.GetMultiText()

 [ ] Clipboard.SetText([LIST OF STRING]lsTempStrings)

 [ ] wMainWindow = TextEditor

 [ ] TextEditor.SetActive()

 [ ] TextEditor.ChildWin("(untitled)
[1]").TextField("#1").SetMultiText(Clipboard.GetText(),2)
```

two_apps.t

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

```
[-] testcase Test1 () appstate DefaultBaseState

 [ ] //SetActive each time you switch apps

 [ ] TestApplication.SetActive()

 [ ] TestApplication.File.New.Pick ()

 [ ] MDIChildWindow1.TextField1.SetPosition (1, 1)

 [ ] MDIChildWindow1.TextField1.TypeKeys ("In Test Application MDI Child
Window #1.")

 [ ] //SetActive each time you switch apps

 [ ] TextEditor.SetActive ()

 [ ] TextEditor.File.New.Pick ()

 [ ] TextEditor.ChildWin("(untitled)[1]").TextField("#1").TypeKeys ("In Text
Editor untitled Document window.<Enter>")

 [ ] //SetActive each time you switch apps

 [ ] TestApplication.SetActive()

 [ ] LIST OF STRING lsTempStrings

 [ ] lsTempStrings = MDIChildWindow1.TextField1.GetMultiText()

 [ ] Clipboard.SetText([LIST OF STRING]lsTempStrings)

 [ ] //SetActive each time you switch apps

 [ ] TextEditor.SetActive()

 [ ] TextEditor.ChildWin("(untitled)
[1]").TextField("#1").SetMultiText(Clipboard.GetText(),2)

 [ ] TextEditor.VerifyCaption("FooBar")

[-] testcase Test2 () appstate DefaultBaseState

 [ ] wMainWindow = TestApplication

 [ ] TestApplication.SetActive()
```

```
 [ ] TestApplication.File.New.Pick ()

 [ ] MDIChildWindow1.TextField1.SetPosition (1, 1)

 [ ] MDIChildWindow1.TextField1.TypeKeys ("In Test Application MDI Child
Window #1.")

 [ ] wMainWindow = TextEditor

 [ ] TextEditor.SetActive ()

 [ ] TextEditor.File.New.Pick ()

 [ ] TextEditor.ChildWin("(untitled)[1]").TextField("#1").TypeKeys ("In Text
Editor untitled Document window.<Enter>")

 [ ] wMainWindow = TestApplication

 [ ] TestApplication.SetActive()

 [ ] LIST OF STRING lsTempStrings

 [ ] lsTempStrings = MDIChildWindow1.TextField1.GetMultiText()

 [ ] Clipboard.SetText([LIST OF STRING]lsTempStrings)

 [ ] wMainWindow = TextEditor

 [ ] TextEditor.SetActive()

 [ ] TextEditor.ChildWin("(untitled)
[1]").TextField("#1").SetMultiText(Clipboard.GetText(),2)
```

Web_Application_Tests

DefaultBaseState closes the Main window in Netscape

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

Loading an HTML page in Netscape brings up a second browser containing an ad. `DefaultBaseState()` closes the main Netscape browser and leaves the ad open. `DefaultBaseState()` closes Browser2 until only one browser remains. If the ad is the active window , then the main browser will be Browser2 and `DefaultBaseState()` will try to close it.

Solution

You may be able to work around the problem by adding the following data member to Browser:

`lwLeaveOpen = (Browser2)`

lwLeaveOpen is part of test frames for client/server applications, but not for browser applications. However, it will work for browsers; you just need to add it manually as follows:

```
[ ]
[ ] // Construct lwLeaveOpen List, if defined either in Browser or wMainWindow
[+] do
[ ] lwLeaveOpenMain Win = wMainWin.lwLeaveOpen
[+] except
```

```
[ ] // lwLeaveOpen not defined for Browser
[ ]
```

File not found error when setting the base state

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

SilkTest displays the Error: `File not found message when setting the base state.`

Solution

Install your user profile for Netscape Navigator in the `<Netscape Installation Directory>\Users \<User name>\` directory.

Ensure that your `<User name>` is truncated to 20 characters and that any spaces in the user name are replaced by underscores.

HtmlPopupList causes the browser to crash when using IE DOM extension

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

Recording or playing back selections against an **HtmlPopupList** causes the browser to crash when using the IE DOM extension.

The problem occurs on browser pages that contain JavaScript. It seems to occur more quickly when recording than when playing back.

Solution

Check the **UseDocumentEvents** option in `extend\domex.ini` and make sure it is set to `FALSE`.

This setting lets you specify whether SilkTest captures OnChange events for an **HtmlPopupList** in order to determine which item was selected. The default is `FALSE`.

Links not being recognized

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

SilkTest is not seeing your links as `HtmlLink` objects.

Cause

You have configured your browser not to underline links. SilkTest requires that links be underlined.

Solution

Reconfigure your browser to display links underlined.

Mouse Coordinate (xy) is off the screen

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

While running the DOM extension, you may receive an error message during playback saying `Mouse Coordinate (x,y) is off the screen`.

Resolution

Clear the following options on the **Verification** tab of the **Options** > **Agent Options** window:

- Verify that windows are exposed
- Verify that coordinates passed to a method are inside the window

This error message may also indicate that the DOM extension was unable to scroll an object into view, for example as part of a `MoveMouse()` call. If that is the case, then set the DOM extension option `UseScrollIntoView` to `TRUE`.

Recording a declaration for a browser page containing many child objects

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When trying to record a declaration for a browser page containing many child objects, CPU usage stays at 100% and SilkTest seems to lock up.

If you add the `Partner.ini` setting:

```
[Runtime]
AgentTimeout=<value in milliseconds>
```

and set it very large, for example `600000` (10 minutes), then eventually SilkTest will return a declaration. The default is `240000` milliseconds (240 seconds or 4 min).

The setting must be put into `Partner.ini`, not into an option set.

For large pages, recording will be very slow, and the CPU will max out while the browser extension initially gathers the page information, and again when it transfers the information to the Agent. If you refrain from moving the mouse around and wait long enough, though, the system will eventually free up.

Recording VerifyProperties() detects BrowserPage properties and children

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

When you record `VerifyProperties()` against the BrowserChild, the recorder misses user-defined properties, and records dynamically instantiated window identifiers of child windows instead of recording the declared window identifiers.

Note that the Record Actions status line detects the window identifier of the BrowserChild. The problem only affects the body of the `VerifyProperties()` method that is recorded.

Solution

If you plan to record `VerifyProperties()` against a declared BrowserChild, then you must include the window declaration for the BrowserChild in a frame (.inc) file that is referenced in the Runtime Options **Use Files** field, before `extend\explorer.inc` or `extend\netscape.inc`.

SilkTest cannot see any children in my browser page in IE 6.x

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

SilkTest does not recognize any children in your application running on Internet Explorer 6.0.

Solution

If you are using IE6.x and only see a BrowserChild with no child objects within it, then make sure that the following option is enabled on the Internet Options dialog in IE6.x:

1. Click **Tools** > **Internet Options**.
2. On the **Internet Options** dialog, click the **Advanced** tab and scroll to the **Browsing** section.
3. Check **Enable third-party browser extensions** (requires restart).
4. Restart your computer.

This IE option is disabled by default in IE6.0.3, which ships with Win2003 Server. Therefore, customers using IE6.0.3 are likely to run into this problem.

SilkTest cannot verify browser extension settings

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

SilkTest is not able to verify extension settings for a Web application running in Internet Explorer.

Cause

Your browser's third-party extensions are not enabled.

Solution

1. In Internet Explorer, click **Tools** > **Internet Options**.
2. Click the **Advanced** tab.
3. Under **Browsing**, select the **Enable third-party browser extensions** (requires restart) check box, then click **OK**.

   In order for the new setting to take effect, you must restart your browser by closing and reopening the browser window.

SilkTest loads an extra Netscape extension

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

SilkTest loads an additional Netscape Navigator extension.

Cause

You have Netscape Navigator Quick Launch enabled.

Solution

Disable Netscape Quick Launch:

1. In Netscape, click **Edit** > **Preferences**.
2. Under **Advanced**, select the **Enable Quick Launch** check box, then click **OK**.

SilkTest not finding Web page of application when you run test on different browser

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

A test you ran successfully on one browser fails on a different browser because SilkTest cannot find the web page of the application.

Cause

Web browsers truncate web page titles if they are too long. Each browser truncates long titles in a different way. SilkTest derives the tag for a page from the page's title. The browser used to record the page declaration will recognize the tag derived from the page's truncated title. Other browsers may not.

The following example shows how Netscape Navigator and Internet Explorer truncate the title of the same page in different ways.

For the Web page entitled `"Heretoday Com Information:`

Internet Explorer	`tag "http:??www.heretoday.com? . . . ?information.htm`
Netscape Navigator	`tag "http:??www.heretoday . . . valuable?information.htm`

Solution

Use a wildcard (*) to abbreviate the tag unambiguously. For example:

```
tag "*information.htm"
```

SilkTest not recognizing web objects

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

When you are recording or playing back tests, SilkTest does not see the objects in your web application. Instead, it sees custom windows.

For example, it might see something like this when you are recording window identifiers and your pointer is on a Web page:

```
MainWin("Netscape*").CustomWin("[Afx:400000:3028]#1")
```

whereas it should be seeing something like this:

```
BrowserPage.HtmlHeading("GMO OnLine")
```

Possible Causes and Solutions

The browser extension is not enabled.	Enable the browser extension.

The browser extension is enabled, but the default browser is not correct. For example, you might have extensions enabled for Explorer and Netscape, but the default browser is Explorer and you are testing Netscape.	Change the browser type. You can change it in the **Runtime Options** dialog or you can specify it in a script using the `SetBrowserType` function.
You are using an older version of the browser.	Upgrade your browser.

SilkTest recognizes static HTML text and tables on a page but does not recognize text such as input fields or bu

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

While using Internet Explorer, SilkTest recognizes static HTML text and tables on a page, but does not recognize text such as HTML fields or HTML buttons.

Cause

You may have improperly nested `<form>` tags, such as the following:

```
<table>
...
<form>
</table>
...
</table>
</form>
```

In the example above, SilkTest does not recognize the second table.

Solution

There are two options:

Check that your HTML tags are properly nested	`<table>`
	`...`
	`</table>`
	`<form>`
	`<table>`
	`...`
	`</table>`
	`</form>`
	
	or
	
	`<form>`
	`<table>`
	`...`
	`</table>`
	`<table>`
	`</table>`
	`</form>`

Set the table recognition value to 0	This means that SilkTest ignores the table and cell, but it will recognize the other input elements.

Test frame containing HTML frame declarations does not compile

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

When you try to compile a test frame that contains several HTML frame declarations, you get one or more "Window is already defined" errors.

Cause

The parent window of your HTML frames is declared more than once.

Solution

For the recommended procedure for recording HTML frame declarations, see *Streamlining HTML frame declarations*.

Two links are merged into a single HtmlLink object (Netscape only)

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

When you enable browser extensions, if you set **Enable JavaScript Always On**, you may find that SilkTest merges two or more distinct links into one `HtmlLink` object or two images into one `HtmlImage` object.

Cause

By default, SilkTest merges adjoining links or images that have the same $Location property value. Setting Enable JavaScript Always On causes the value of $Location to always be NULL. SilkTest interprets this condition to mean that the links or images have the same $Location value and merges the links or images into one object.

Solution

Disable the merging for the duration of the Netscape or SilkTest Agent session. In a script, call `SetUserOption` method as follows:

```
BrowserPage.SetUserOption("EmptyTextLocationsDiffer", TRUE, USEROPT_DEFAULT)
BrowserPage.SetUserOption("EmptyImageLocationsDiffer", TRUE, USEROPT_DEFAULT)
```

These calls cause SilkTest to report a single link as two separate links if the link's text is displayed on two lines.

Web property sets not displayed during verification

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

The **Verify Window** dialog is not displaying the property sets for the browser extensions (Color, Font, Values, and Location).

Possible Causes and Solutions

No browser extension is enabled.	Make sure that at least one browser extension is enabled.
Enhanced support for Visual Basic is enabled.	Disable Visual Basic by unchecking the ActiveX check box for the Visual Basic application in the Extension Enabler dialog and the Extensions dialog.

Why does the SilkTest recorder generate so many MoveMouse() calls

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you are recording against a page that contains DHTML popup menus or other elements with JavaScript mouse movement event handlers (such as `onmouseover`), you will notice that many `MoveMouse()` calls are generated. The `MoveMouse()` calls are recorded in order to improve reliability of playback against DHTML popup menus by ensuring that SilkTest exposes the menus as it navigates through them. This behavior was added in SilkTest 7.1 as a replacement for the previous requirement of holding down the **<Shift>** key while recording against DHTML popup menus, since that requirement would not have been easily known to users. There is no straightforward way for SilkTest to know whether or not a mouse movement event handler belongs to an element (such as a popup menu) that requires exposure. Therefore `MoveMouse()` is recorded for any such element in order to err on the side of caution and ensure that the event handlers are triggered during playback.

There is no setting that directs the recorder to exclude the `MoveMouse()` calls. The extra calls should not cause problems or slow playback, but if you are bothered by them, then they can be deleted manually from the scripts (as long as the target elements do not require exposure).

Objects

Does SilkTest support Oracle Forms

This functionality is available only for projects or scripts that use the Classic Agent.

SilkTest supports Oracle Forms applications as it would any Java applet that consists of custom classes.

All children of the applet are seen as `CustomWins`, with native class names such as 'oracle.ewt.*' and 'oracle.forms.*'. You need to declare winclasses for any classes that you plan to use, and you can only interact with classes through scripting. For more efficient declaration of classes, use the `CaptureAllClasses`() function instead of clicking **Record** > **Class** to record each class separately.

As with any application consisting of custom classes, if there are objects that SilkTest does not see, check **Show All Classes** to see if that exposes the ignored objects. If so, then you should add those classes to the `[ClassList]` section of `extend\JavaEx.ini`. Uncheck **Show All Classes** before recording window classes or declarations.

To get started, take a look at our guidelines for when and how to record classes.

If you do not want to record classes for these `CustomWin` objects, you can click **Record** > **Class** and then uncheck the **Show All Classes** check box in the lower left corner of the dialog box.

Mouse Clicks Fail on Certain JFC and Visual Café Objects

This functionality is available only for projects or scripts that use the Classic Agent.

Because of timing issues between the Agent and the application under test, you might experience problems with menu picks on Java Foundation Class (JFC) or Symantec Visual Café objects. As a workaround, try setting keyboard and mouse delays. Use the values specified below as good starting points, and then experiment with different delays as needed until you find the timing that works best for your system configuration and application.

For ...	Set ...
Java Foundation Class objects	keyboard delay = 0.01 second mouse delay = 0.01 second
Visual Café objects	keyboard delay = 0.03 second mouse delay = 0.03 second

My Sub-Menus of a Java Menu are being Recorded as JavaDialogBoxes

This functionality is available only for projects or scripts that use the Classic Agent.

Occasionally, a sub-menu of a Java menu exceeds the boundaries of its application so that part of the sub-menu extends beyond the borders of the application. In this situation, SilkTest records these very large sub-menus as `JavaDialogBoxes`, not as part of the menu.

Try dragging the `JavaMainWin` to a larger size before recording or maximizing the application.

Other Problems

Application hangs when playing back a menu item pick

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

Your application under test hangs when playing back a `Pick()` method call against a menu item (or menu).

Solution

Try setting the Agent option `OPT_PLAY_MODE` to `Win32`:

```
Agent.SetOption (OPT_PLAY_MODE, "Win32")
```

This option is not part of the **Agent Options** dialog, so you must set it by scripting. To set it globally, create a `TestCaseEnter()` function and set it there.

For more information on this option, see the description of `OPT_PLAY_MODE`.

Cannot access some of the SilkTest menu commands

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

You cannot use several of the menus/menu commands.

Solution

You may be using **SilkTestRuntime**, a stripped down version of SilkTest. To check what version you are using, click **Help** > **About**.

Cannot double click a SilkTest file and open SilkTest

Problem

SilkTest does not open automatically when you double-click a `.t`, `.inc`, `.s`, `.g.t`, `.pln`, `.res`, `.stp`, or `.vtp` file.

Cause

During the install process, SilkTest is associated with these file types. However if these file type associations have been changed after SilkTest setup, these file types may not be opened with SilkTest when double-clicking such a file.

Note: File type associations are only available for Microsoft Windows platforms.

Solution

You can either manually associate these file types with SilkTest in Windows (**Start** > **Settings** > **Control Panel** > **Folder Options**) or reinstall SilkTest.

Cannot extend AnyWin Control or MoveableWin classes

The `AnyWin`, `Control`, and `MoveableWin` classes are logical (virtual) classes that do not correspond to any actual GUI objects, but instead define methods common to the classes that derive from them. This means that SilkTest never records a declaration that has one of these classes.

Furthermore, you cannot extend or override logical classes. If you try to extend a logical class, by adding a method, property or data member to it, that method, property, or data member is not inherited by classes derived from the class. You will get a compilation error saying that the method, property, data member is not defined for the window that tries to call it.

Nor can you override the class, by rewriting existing methods, properties, or data members. Your modifications are not inherited by classes derived from the class.

Cannot find the Quick Start Wizard

This functionality is available only for projects or scripts that use the *SilkTest Classic Agent*.

Problem

You cannot find the Quick Start Wizard.

Solution

Beginning with SilkTest 6.0, the Quick Start Wizard is turned off by default. You can turn the Wizard back on by opening `partner.ini` and replacing these two lines:

```
[Wizard]
AutoInitWizard=FALSE
```

With the following:

```
[Wizard]
AutoInitWizard=TRUE
WizardEnabled=TRUE
```

Restart SilkTest. The Wizard is available when you click **File** > **New** > **Testframe**.

Cannot play back menu picks against Java application

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

Using Netscape, you cannot play back menu picks against a Java application launched from an applet.

You cannot currently test Java applications launched from applets using SilkTest.

Solution

There are two possible workarounds for the problem.

1. When using Microsoft's JVM (`jview.exe`), it is necessary to set an Agent option in each test case to be run:

   ```
   Agent.SetOption (OPT_MENU_SELECT_TOPPOPUP, "<Button1>")
   ```

   Instead of including the Agent option in each testcase, you can create a `TestCaseEnter` function and include the Agent option in the function.
2. Use **Win32** playback mode:

   ```
   Agent.SetOption (OPT_PLAY_MODE, "Win32")
   ```

Cannot play back picks of cascaded submenus for an AWT application.

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

For AWT, SilkTest can pick cascaded menu items only up to the third level, for example: `TestApplication.Menu.TheCascadeItem.Su\menu1.Item2.`

It cannot pick deeper submenus, such as `TestApplication.Menu.TheCascadeItem.Su \menu1.Submenu2.Item2.`

For JVM 1.1.x, SilkTest may only be able to pick menu items up to the 2nd level, for example: `TestApplication.Menu.TheCascadeItem.It\m2.`

Cannot record second window

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

When SilkTest records a popup list or listbox select that causes an `OnMouseDown` event to spawn a second window, it cannot record that second window. This is usually due to the Javascript that launches a second window and changes the underlying value of the popuplist or listbox.

You must edit the recorded scripts in order for play back to run correctly.

Ignoring a Java class

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

If you are using the Java extension and you want to ignore a class, you must edit your `javaex.ini` file. Add the following line to your `javaex.ini` file:

`myclass=FALSE` (where `myclass` is the full class name of the class to be ignored).

Use the value `FALSE` and not the value `IGNORE`.

Cannot open results file

Problem

SilkTest crashes while running a script and reports the error `Can't open results file`.

Solution

While SilkTest is running a script, it temporarily stores results in a journal file (`.jou`) which is converted to a `.res` file when the script finishes running.

Delete all `.jou` files in the same directory as the script. (You do not have to delete your results files.)

Restart SilkTest and run your script again.

Common DLL problems

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Here are some issues that could come up if you are calling DLL functions in a script.

`Error in results file: dll not found`

This usually means that your path does not include the directory containing the DLL. If you are running remotely, make sure that the path on the machine running the Agent includes the DLL directory.

`Error after compile: dll not found`

In the DLL declaration, use the fully qualified path of the DLL, not just the file name.

`Error in results file: function <name> not found in dll`

The most likely scenario is that the DLL is a C++ library and the function name has been mangled. To use functions in a C++ library, you need to wrap the functions with the C wrapper and recompile. Then SilkTest can access the function in the library.

If this is not the problem, there might be a typo in the function name in the DLL.

`Error in results file: dll could not be loaded`

Make sure the directory containing the DLL is on the path.

`Warning in results file: String buffer size was increased from x to 256 characters`

If the user calls a DLL function with an output string buffer that is less then the minimum size of 256 characters, the original string buffer is resized to 256 characters and a warning is printed. This warning,

`String buffer size was increased from x to 256 characters` (where x is the length of the given string plus one) alerts the user to a potential problem where the buffer used might be shorter than necessary.

`Difficulty creating DLLs to use with SilkTest`

Only specific data types are compatible with 4Test. These data types are listed in *C data types for DLL functions*.

If your DLL calls have data types not supported by 4Test, then the functions must be wrapped such that only compatible data types are used for the return type and arguments of the function. Any data types can be used inside the DLL function.

Common scripting problems

Here are some common problems that occur with scripts.

Typographical errors

It is very easy to make typographical errors that the 4Test compiler cannot catch. If a line of code does nothing, this might be the problem.

Global variables with unexpected values

When you write a function that uses global variables, make sure that each variable has an appropriate value when the function exits. If another function uses the same variable later, and it has an unexpected value on entry to the function, an error could occur.

To check that a variable has a reasonable value on entry to a function, set a breakpoint on the line that calls the function and use the command View/Global Variables to check the variable's value.

Uninitialized variables

SilkTest does not initialize variables for you. So if you have not initialized a variable on entry to a function, it will have the value `<unset>`. It is better to explicitly give a value to a variable than to trust that another function has already initialized it for you. Also, remember that 4Test does not keep local variables around after a function exits; the next time the function is called, its local variables could be uninitialized.

If you are in doubt about whether a variable has a reasonable value at a particular point, set a breakpoint there and use **View** > **Global Variables** or **View** > **Local Variables** to check the variable's value.

Global and local variables with the same name

It is usually not good programming practice to give different variables the same names. If a global and local variable with the same name are in scope (accessible) at the same time, your code can only access the local variable.

To check for repeated names, use **View** > **Local Variables** and **View** > **Global Variables** to see if two variables with the same name are in scope simultaneously.

Incorrect values for loop variables

When you write a for loop or a while loop, be sure that the initial, final, and step values for the variable that controls the loop are correct. Incrementing a loop variable one time more or less than you really want is a common source of errors.

To make sure a control loop works as you expect, use **Debug** > **Step** Into to step through the execution of the loop one statement at a time, and watch how the value of the loop variable changes using **View** > **Local Variables**.

Checking the precedence of operators

The order in which 4Test applies operators when it evaluates an expression may not be what you expect. Use parentheses, or break an expression down into intermediate steps, to make sure it works as expected. You can use View/Expression to evaluate an expression and check the result.

Incorrect uses of break statements

A break statement transfers control of the script out of the innermost nested for, for each, while, switch, or select statement only. In other words, break exits from a single loop level, not from multiple levels. Use **Debug** > **Step** Into to step through the script one line at a time and ensure that the flow of control works as you expect.

Infinite loops

To check for infinite loops, step through the script with **Debug** > **Step Into**.

Code that never executes

To check for code that never executes, step through the script with **Debug** > **Step Into**.

Conflict with virus detectors

Problem

SilkTest will occasionally have problems on machines running virus detectors that use heuristic or algorithmic virus detection in addition to the standard pattern recognition. What happens is that while SilkTest is running, the virus detector identifies SilkTest as displaying "virus-like" behavior, and kills or otherwise disables the Agent. This leads to unpredictable and inconsistent behavior in SilkTest including loss of communications with the agent and inconsistent test results and/or object recognition.

Solution

To avoid this problem the only solution is to temporarily disable the virus detector while SilkTest is running.

Do I need administrator privileges to run SilkTest

You must have local administrator privileges to install SilkTest on a Windows 2000 or XP machine. Once SilkTest is installed, you must also have administrator privileges in order to run SilkTest. If you are installing on a Windows 2000 or XP server, you must have domain-level administrator privileges.

General Protection Faults

Problem

When recording or running tests, you get a `General Protection Fault (GPF)` or `Invalid Page Fault (IPF)` in `agent.exe` or `partner.exe`.

Solution

It can be very difficult to pin down the cause of these problems. It might involve a combination of your machine's configuration, other applications that are running, and the network's configuration. The best approach is to gather the diagnostic information described below and send it to Technical Support with a detailed description of what scenario led to the error.

Capture the system diagnostics	When the system error message displays, chose the option to capture detailed information on the error. Write the information down.
Capture a debug.log file	**1.** Ensure that no SilkTest or Agent processes are running. **2.** Open a DOS prompt window. **3.** Change your working directory to your SilkTest installation directory. **4.** Delete or rename `c:\debug.log` if the file exists. **5.** Set the following environment variable: `set QAP_DEBUG_AGENT=1`. **6.** Start the Agent manually: `start .\agent`. **7.** Start SilkTest manually: start `.\partner`. **8.** Go through the scenario to reproduce the problem. **9.** The file `c:\debug.log` file will be created. **10.** Send this file as an attachment to your email to Technical Support.
Monitor CPU and RAM usage	When reproducing this error to gather the diagnostics above, also run a system resource monitor to check on CPU and RAM usage. Note whether CPU or RAM is being exhausted.
Note your system configuration	When sending in these diagnostics, note the version of SilkTest, the operating system and version, and the machine configuration (CPU, RAM, disk space).

Include file or script compiles but changes not picked up

Problem

You compile an include file or script, but changes that you made are not used when you run the script.

Solutions

Did you change the wrong include file?	Make sure that the include file you are compiling is the same as the file that is being used by the script. Just because you have an include file open and have just compiled it does not mean that it is being used by the script. The include file that the script will use is either specified in Runtime Options (Use Files field) or by a use statement in the script.
Is there a time-stamp problem?	If the time stamp for the file on disk is later than the machine time when you do **Run** > **Compile**, then the compile does not actually happen and no message is given. This can happen if two machines are sharing a file system where the files are being written out and the time on the machines is not synchronized. By default, SilkTest only compiles files that need compiling, based on the date of the existing object files and the system clock. This way, you don't have to wait to recompile all files each time a change is made to one file. If you need to, you can force SilkTest to compile all files by selecting **Run** > **Compile All**. **Run** > **Compile All** compiles the script or suite and all dependent include files, even if they have not changed since they were last compiled. It also compiles files listed in the **Use Files** field in the **Runtime Options** dialog and the compiler constants declared in the **Runtime Options** dialog. Finally, it compiles the include files loaded at startup, if needed.

Are your object files corrupted?

Sometimes a SilkTest object (`.ino` or `.to`) file can become corrupted. Sometimes a corrupted object file can cause SilkTest to assume that the existing compile is up to date and to skip the recompile without any message.

To work around this, delete all `.ino` and `.to` files in the directories containing the `.inc` and `.t` files you are trying to compile, then compile again.

Library Browser not displaying user-defined methods

Problem

You add a description for a user-defined method and a user-defined function to `4test.txt`. After restarting SilkTest, the new description for the function displays in the Library Browser, but not the description for the method. So you know that the modified `4test.txt` file is being used, but your user-defined method is not being displayed in the **Library Browser**.

Solutions

Only methods defined in a class definition (that is, in your include file where your class is defined) will display in the **Library Browser**. For example, `MyAccept` will be displayed.

```
winclass DialogBox:DialogBox
Boolean MyAccept()
...
```

Methods you define for an individual object are not displayed in the **Library Browser**. For example, `MyDialogAccept` will not display.

```
DialogBox MyDialog
tag "My Dialog"
Boolean MyDialogAccept()
...
```

In order to display in the **Library Browser**, the description in your `4test.txt` file must have a return type that matches the return type in your include file declaration. If the `4test.txt` description has no `returns` statement, then the declaration must be for a return type of void (either specified explicitly or by defaulting to type void). Otherwise, the description will not display in the **Library Browser**.

For more information about adding information to the **Library Browser**, see *Adding to the Library Browser*.

Global variables Running from a testplan versus running from a script

Problem

When running from a testplan, global variables don’t keep their value from one testcase to another.

When testcases are run from a script, global variables are initialized once at the beginning and do not get reset while the script is being run. On the other hand, when you run testcases from a testplan, all global variables get re-initialized after each testcase. This is because the Agent reinitializes itself before running each testcase. Consequently, you may find that global variables are not as useful when running from a testplan.

Solution

A workaround is to use the `FileWriteLine` or `FileWriteValue` function to write the values of the global variables out to a file, then use the `FileReadLine` or `FileReadValue` function to read the value back into each variable in each testcase.

Maximum size of SilkTest files

The following size limits apply:

- The limit for `.inc`, `.t`, and `.pln` files (and their associated backup files, `.*_`) is 64K lines.
- The size limit for the corresponding object files (`.*o`) depends on the amount of available system memory.
- The SilkTest editor limits lines to 1024 characters.
- The maximum size of a single entry in a .res file is 64K.
- Testcase names can have a maximum of 127 characters. When you create a data driven testcase, SilkTest truncates any testcase name that is greater than 124 characters.

Playing Back Mouse Actions

This functionality is available only for projects or scripts that use the Classic Agent.

Under 32-bit Windows, the following methods take an optional BOOLEAN argument, `bRawEvent`, that specifies how mouse actions are played back:

AnyWin methods	• Click • DoubleClick • MoveMouse • MultiClick • PressMouse • ReleaseMouse
Pushbutton method	• Click

By default, `bRawEvent` is FALSE. When FALSE, SilkTest uses the standard Windows messaging mechanism (journal playback) to perform actions. Usually this works fine. If your test plays back correctly, use the default.

There are times, however, when this doesn't work and your test won't play back correctly. In such situations, set `bRawEvent` to `TRUE`. When `TRUE`, SilkTest uses a low-level mechanism to perform the actions. Operations involving mouse dragging are more likely to work correctly using the low-level mechanism. But this mechanism hasn't been tested as thoroughly as journal playback, so you should use it only when the default fails.

You can have all playback use the low-level mechanism by setting OPT_PLAY_MODE to Win32:

```
Agent.SetOption (OPT_PLAY_MODE, "Win32")
```

To turn this off, set OPT_PLAY_MODE to Normal:

```
Agent.SetOption (OPT_PLAY_MODE, "Normal")
```

Recorder does not capture all actions

Problem

While recording, the recorder does not capture all actions in your application under test, though you complete the actions.

Cause

The application under test may be "going too fast" and the SilkTest Recorder may not be able to keep up.

Solution

Slow down the interactions with your application while recording. Record a testcase at the speed of the Recorder.

Recording two SetText () statements

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

While using the Record Actions dialog, you capture entering `John` into a text field. SilkTest may record the following statements:

```
<identifier>.SetText ("J")
<identifier>.SetText ("John")
```

This is not an error. The recorder may capture several `SetText` statements without impacting playback.

Relationship between exceptions defined in 4test.inc and messages sent to the result file

SilkTest calls LogError automatically when it raises an exception that you have not handled. By reading `4test.inc` you can find that Silk has a list of exceptions like:

```
E_ABORT = -10100,
E_TBL_HAS_NO_ROW_HDR = -30100,
E_WINDOW_NOT_FOUND = -27800
```

Since exception numbers can apply to more than one exception, it can be helpful to query on a particular exception number via `ExceptNum()` to decide how to handle an error. If you need to query on a specific exception message, you can use `ExceptData()`. We recommend using `MatchStr()` with `ExceptData()`.

To find the `E_...` constant for any 4Test exception, you can use:

```
[-] do
    <code that causes exception>
[-] except
[ ] LogWarning ("Exception number: {[EXCEPTION]ExceptNum ()}")
[ ] reraise
```

This will print out the exception constant in the warning.

Be sure to remove the `LogWarning do..except` block after you have found the `E_...` constant.

SilkTest 4test editor does not display enough characters

Problem

While you can edit 4test files outside of SilkTest and create lines with more than 1024 characters, the SilkTest editor does not let you edit or extend them.

The SilkTest editor's line limit is 1024 characters.

Solution

Use the `<Shift+Enter>` continuation character to break the line into smaller lines.

SilkTest support of Delphi applications

While there is no support for Delphi controls "out of the box", virtually all of the Delphi objects can be class mapped to standard controls.

```
[ClassMap]
DialogBox,0x50000044,0x50000044=Ignore
TBitBtn=PushButton
TButton=PushButton
TCheckBox=CheckBox
TComboBox=ComboBox
TDBCheckBox=CheckBox
TDBComboBox=ComboBox
TDBEdit=TextField
TDBListBox=ListBox
TDBLookupComboBox=ComboBox
TDBLookupListBox=ListBox
TDBMemo=TextField
TDBRadioGroup=Ignore
TEdit=TextField
TFlyingPanel=Toolbar
TGroupBox=StaticText
TGroupButton=RadioButton
TListBox=ListBox
TListView=ListView
TMaskEdit=TextField
TMemo=TextField
TPageControl=PageList
TPanel=Ignore
TRadioButton=RadioButton
TRadioGroup=Ignore
TRichEdit=TextField
TRicherEdit=TextField
TScrollBar=ScrollBar
TStatusBar=StatusBar
TTabControl=PageList
TTreeView=TreeView
TUpDown=UpDown
```

There are a few custom controls that don't respond to class mapping (such as TreeViews, Grids, and StaticText). However, `TypeKeys` and mouse events can be used on them.

`GetContents()` seems to return nothing for TreeViews.

Notes

SilkTest can work with Delphi objects in a variety of ways. The amount of functionality you achieve depends on how deep you want to get involved. You can even create an extension (external) for Delphi objects. Delphi supports DLL calling, and you can use DLL's created in C/C++ in your Delphi application. Class mapping will work in many instances, but not with every object.

If class mapping doesn't work, you can try any of the following workarounds:

1. Using `SendMessage` with the Clipboard

   - Delphi is built with VCL. The VCL (Visual Component Library) is similar to MFC in that all of the classes of objects that Delphi can create are in this library. Instead of C++ it is written in Object Pascal. We ship the VCL source code with the product. In the VCL source, you can go to the definition of the object class that you want to support for and add message handlers (windows API messages) for various messages that you define.
   - For example, add a message handler that says that if any object of this class receives a message called `QAP_GetValue`, get the contents of the listbox, send a message back to the process that

sent the message, and send it the value. On the SilkTest side of things you define a new class to support the object and add a method that sends/receives the message to the supported object.

- For example, here is sample code of a message handler on the Delphi side:

```
procedure QAP_GetValue (var Msg: TMessageRecord);
var
ValueToReturn : string;
begin
CopyToClipBoard;
Msg.Result := true;
end;
```

- Here is sample code for the `Window` class on the 4Test side:

```
winclass DelphObj : Control
LIST OF STRING GetContents ()
if (SendMessage (this.hWnd, QAP_GetListContents, NULL, NULL))
return Clipboard.GetText ()
else
RaiseError (1, "Couldn't get the contents of {this},
SendMessageEvent not processed correctly")
```

2. Using the Extension Kit, create a DLL that does the same thing as above, except passing values directly from application to application rather than relying on the clipboard. This method is preferred over the above because of speed and data type stability.
3. Use low level 4Test events relying on coordinates to create methods. SilkTest low-level recording should only be used when you want to use recording rather than hand scripting.

Stopping a testplan

Problem

You want to abort a testplan programmatically without using `exit`. Calling `exit` just aborts the script and continues on to the next testcase.

Solution

You can call: `[ ] @("$StopRunning") ()`

from a testcase or a recovery system function such as `ScriptExit()` (which is called for each testcase in the testplan) or `TestCaseExit()`.

That call will stop everything without even invoking the recovery system. Calling it will generate the exception message (with no call stack):

```
Exception -200000
```

TextField not allowing input

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Problem

A TextField doesn't accept input from `TypeKeys` and `SetText` or allow a paste from the **Clipboard**.

For example, in the following script, the **Password** TextField doesn't get any text set in it.

```
EnterNetworkPassword.SetActive ()
EnterNetworkPassword.Password.SetText ("mypassword")
EnterNetworkPassword.OK.Click ()
```

Solution

Make a DLL call to `SendMessage` (declared in `msw32.inc`) in the following way:

```
use "msw32.inc"
...
Clipboard.SetText ({"mypassword"})
EnterNetworkPassword.Password.DoubleClick ()
SendMessage (EnterNetworkPassword.Password.hWnd,WM_PASTE, 0,0)
```

By using the API message `WM_PASTE` in a `SendMessage` call, the TextField will get populated with the text that is on the **Clipboard**.

Adding a property to the recorder

1. Write a method.
2. Add a property to the class.
3. Add the property to the list of property names.

For example, if you have a text field that is `ReadOnly` and you want to add that property to the recorder you can do the following:

1. Write the method `Boolean IsReadOnly()` for the `TextField` class.
2. Add the property, `bReadOnly` to the class.
3. Add `bReadOnly` to the list of property names.
4. Compile. `bReadOnly` will appear in the **Recorder** after you compile.

```
Winclass TextField : TextFieldBOOLEAN IsReadOnly()
STRING sOriginalText = this.GetText()
STRING sNewText = "xxx"
this.SetText(sNewText)
if this.GetText()==sOriginalText
return TRUE
else
return FALSE
property bReadOnly
BOOLEAN Get()
return this.IsReadOnly()
LIST OF STRING IsPropertyNames = {…}
"bReadOnly"
```

Displaying the Euro symbol in SilkTest

Problem

You want to display the Euro symbol.

Solution

Download a Euro-enabled font from Microsoft. Double check that you can see the Euro symbol by opening Notepad on the machine where you installed the font and entering the ASCII code for the Euro symbol. As long as you see the symbol in notepad, you should be able to see it within SilkTest.

In SilkTest, click **Options** > **Editor Font** and be sure that your font is set to Arial, Courier New, or Times New Roman.

Improving performance when Verifying Properties

On Win2000, you can improve performance when you are verifying properties by setting or editing a registry value as follows:

1. Locate the `fglockto.reg` file on your SilkTest installation CD in `\SilkTest\W98_2000`.
2. Double-click the `fglockto.reg` file and follow the directions to set or edit a registry value.
3. Reboot after you run this file.

Running this file improves issues shifting focus from SilkTest to the browser.

Using SilkTest file functions to add information to the beginning of a file

As of SilkTest 5.5 SP1, there is no file open mode that allows you to insert information into the beginning of a file. If you use `FM_UPDATE`, you can read in part of your file before writing, but any write function calls will overwrite the rest of the file.

If you are writing strings rather than structured data, you can use `ListRead()` and `ListWrite()` to insert information at the beginning (or any other point) of a file. Use `ListRead()` to read the contents of the file into a list, insert the new information at the head (or any other point) of the list, and use `ListWrite()` to write it back out.

```
[-] LIST OF STRING lsNewInfo = {...}
[ ] "*New line one*"
[ ] "*New line two*"
[ ] "*New line three*"
[ ] LIST OF STRING lsFile
[ ] INTEGER i
[ ]
[ ] ListRead (lsFile, "{GetProgramDir ()}\Sample.txt")
[-] for i = 1 to ListCount (lsNewInfo)
[ ] ListInsert (lsFile, i, lsNewInfo[i])
[ ] ListWrite (lsFile, "{GetProgramDir ()}\Sample.txt")
[ ]
```

`Sample.txt` before:

```
Line 1
Line 2
Line 3
Line 4
Line 5
```

`Sample.txt` after:

```
*New line one*
*New line two*
*New line three*
Line 1
Line 2
Line 3
Line 4
Line 5
```

Recognition Issues

I Cannot See all Objects in my Application even after Enabling Show All Classes

This functionality is available only for projects or scripts that use the Classic Agent.

If some of the objects in your application are derived from the AWT Object class, instead of the AWT Component Class, SilkTest will not be able to find them. In this situation, we recommend using the `invokeJava` method to manipulate these objects.

java.lang.UnsatisfiedLinkError

This functionality is available only for projects or scripts that use the Classic Agent.

When recording window declarations, the following error displays:

```
SilkTest Java extension loaded, running under JDK version x
java.lang.UnsatisfiedLinkError: no qapjarex in java.library.path
```

Copy the `qapjarex.dll` from the `System32` directory into the `lib\ext` directory of the JRE installed by the application.

My JavaMainWin is Not Recognized

This functionality is available only for projects or scripts that use the Classic Agent.

If SilkTest sees the main window of your Java application or applet as `MainWin` instead of `JavaMainWin`, or recognizes the main window, but none of its child controls, you can run a Java status utility to help you or Technical Support diagnose the problem.

One common cause is that SilkTest is not properly configured to test Java.

If you are running an applet using the plug-in for JVM 1.4+, SilkTest may recognize the main window as class `DialogBox`, not `JavaMainWin`, and will not see any child objects. The solution is to class-map the top-level class of the applet to `JavaMainWin`. For example:

```
microfocus.com.appclass=JavaMainWin
```

None of My Java Controls are Recognized

This functionality is available only for projects or scripts that use the Classic Agent.

If you notice that SilkTest does not recognize any of your Java controls, which means that SilkTest sees them all as `CustomWin` objects, make sure you have set up your test environment correctly by:

1. Ensuring that you have configured SilkTest support for Java correctly.
2. Ensuring that you have enabled the Java extension correctly, based on the runtime environment your Java application invokes, as explained in the following table:

If your application invokes:	Enable Java support by:
• `java.exe` (Java Development Kit) • `jre.exe` (Java Runtime Environment, standard version that invokes a console window) • `jrew.exe` (Java Runtime Environment, version that does not invoke a console window) • `vcafe.exe` (Symantec Visual Café 2.0) • `appletviewer.exe`	Enabling the Java extension for **Java Application** in the **Extension Options** dialog box.
Any other runtime environment that uses a different executable or dll.	Adding the .exe or .dll file as an application in the **Extension Enabler** and **Extension Options** dialog box. Enabling the Java extension for this application in the **Extension Enabler** and **Extension Options** dialog box.

Only JavaMainWin

This functionality is available only for projects or scripts that use the Classic Agent.

If during recording of a window declaration, SilkTest sees the main window as a `JavaMainWin` or `JavaDialogBox`, but does not see any child objects, make sure that your classpath references the correct SilkTest .jar file.

If your application sets the Java library path using the JVM launcher directive `Djava.library.path=<path>`, you must copy `qapjarex.dll` from the `System32` directory into the location pointed to by the JVM launcher directive. SilkTest should then recognize child objects.

Only Applet Seen

This functionality is available only for projects or scripts that use the Classic Agent.

If only the Java applet is seen during the recording of a window declaration, and no other objects are recognized, check the following:

- The Java extension is enabled.
- Your classpath references the correct SilkTest .jar file.

SilkTest Does not Record Click() Actions Against Custom Controls in Java Applets

This functionality is available only for projects or scripts that use the Classic Agent.

Security restrictions may prevent SilkTest from recording `Click`() actions against custom controls in Java applets. This may happen if the applet is from an untrusted source.

Verify that you have the correct security permissions set up for your AUT. The following steps show how you can verify the settings for the test applet, installed in the `<SilkTest installation directory>/javaex/jfc11` directory:

1. Locate the plug-in used to run the Applet. If you have multiple plug-ins installed, you may find the used plug-in with the java console running(use the plug-in setting Show Console). To display the system properties with java.home pointing to the directory of used plug-in, type "s" while the console is active.

   This is usually located in the `program files/java/jre_name` directory. If you have multiple plug-ins installed, you may find the used plug-in with the java console running. Use the plug-in setting **Show Console**. To display the system properties with `java.home` pointing to the directory of used plug-in, type **s** while the console is active.
2. Locate the `lib/security` directory located under the plug-in directory.
3. Open the `java.policy` file. Verify that the following fragment is in the file, or add it to the file, if it is not.

   This assumes that you have installed SilkTest into the default installation directory, `C:\program files\Silk\silktest`; if you installed SilkTest into a different directory, you must change the fragment accordingly.

```
grant codeBase "file:C:/program files/silk/silktest/javaex/jfc11/*"
{
  permission java.security.AllPermission;
};
```

Note: The file protocol is used here because the applet is located on the host machine. If the applet was downloaded from a URL instead, then you must substitute the appropriate http://url_name instead.

SilkTest Does not Recognize a Popup Dialog Box caused by an AWT Applet in a Browser

This functionality is available only for projects or scripts that use the Classic Agent.

If an AWT applet in a browser causes a popup dialog box to appear, SilkTest does not see it as a `JavaDialogBox` and does not see any of the controls within the dialog box.

Click **OptionsClass Map** to map the `AppletPopup` custom class to the `JavaDialogBox` class.

SilkTest is not recognizing updates on Internet Explorer page containing JavaScript

This functionality is available only for projects or scripts that use the Classic Agent.

If SilkTest does not recognize updates made to an Internet Explorer page containing JavaScript, SilkTest does not know that the page changed because the existing objects change, but nothing gets created or destroyed.

In such a case, call `BrowserPage.FlushCache()` in between the update methods. The `FlushCache` method is useful when a JavaScript event causes an update to existing objects on the page, but does not cause any objects to be created or removed.

Some of My Java Controls are Not Recognized

This functionality is available only for projects or scripts that use the Classic Agent.

By default, SilkTest ignores objects that are usually not relevant for testing, such as containers and panels, to promote efficient recording. In some situations, however, user-defined objects or third-party JavaBeans might also be ignored inadvertently.

You can access these objects for testing by recording classes for ignored objects in standalone Java applications or in Java applets.

Verify Properties does not Capture Window Properties

This functionality is available only for projects or scripts that use the Classic Agent.

If **Verify Properties** does not capture window properties for a stand-alone Java application, do not position the cursor in title bar to verify properties.

In order to use **Verify Properties** against a stand-alone Java application, position your cursor at a point within the client area of the window. Do not position the cursor in the title bar, as that may prevent SilkTest from being able to capture the window properties.

Tips

Troubleshooting Tips

This topic contains information that can help you if you run into problems.

See the Release Notes by choosing **Start** > **Programs** > **Silk** > **SilkTest <version>** > **Release Notes** for the most current information about troubleshooting and workarounds.

Help on error messages

Agent not responding

BrowserChild MainWindow Not Found When Using Internet Explorer 7.x

Cannot find file agent.exe

Control is not responding

Unable to start Internet Explorer

Variable Browser not defined

Variable not defined

Window Browser does not define a tag

Window is not active

Window is not enabled

Window is not exposed

Window not found

Help on Web application tests

DefaultBaseState closes the Main window when testing browsers

File not found error when setting the base state

HtmlPopupList causes the browser to crash when using IE DOM extension

Links not recognized

Mouse Coordinate (x,y) is off the screen

Recording VerifyProperties() detects BrowserPage properties and children

SilkTest cannot see any children in my browser page in IE6.x

SilkTest cannot verify browser extension settings

SilkTest does not recognize text such as input fields or buttons if the <form> tag is not properly nested

SilkTest loads an extra Netscape extension

Test frame containing HTML frame declarations does not compile

Two links are merged into a single HtmlLink object (Netscape only)

Web classes not displayed in Library Browser

Web objects not recognized

Web page of application not found when you run test on different browser

Web property sets not displayed during verification

Why does the SilkTest recorder generate so many MoveMouse() calls?

Help on working with projects

Editing the projectname.vtp and projectname.ini files

I can't find items in Classic 4Test

My files no longer appear in the Recently Opened Files list

SilkTest cannot find a file when autogenerating a new project

SilkTest cannot find files when opening my project

SilkTest cannot load my project file; the contents of my projectname.vtp have changed or my projectname.ini file has been moved

SilkTest cannot save files to my project

SilkTest did not add my .inc file when autogenerating my project

SilkTest won't run

Help on other problems

ActiveX/Visual Basic controls

Application hangs when playing back a menu item pick

Cannot access some of the SilkTest menu commands

Cannot double click a SilkTest file and automatically open SilkTest

Cannot extend AnyWin, Control, or MoveableWin classes

Cannot find the Quick Start Wizard

Cannot play back menu picks against Java app

Cannot play back picks of cascaded submenus for an AWT application

Cannot record second window

Can't ignore a Java class

Can't open results file

Common problems in scripts

Common problems calling DLLs

Correcting problems when using Java support

Conflict with virus detectors

Do I need administrator privileges to run SilkTest?

Enhancing recovery system to handle multiple applications

General Protection Faults

Include file or script compiles but changes not picked up

Library Browser not displaying user-defined methods

Library Browser not displaying Web browser classes

Link Tester does not run from SilkTest menu

Recorder does not capture all actions

Recording two SetText () statements

SilkTest 4test editor does not display enough characters

SilkTest doesn't launch my Java Web Start application

Stopping a testplan without using "exit"

TextField not allowing input

Using global variables in a testplan

Hints

Adding a property to the recorder

Displaying the Euro symbol in SilkTest

Improving performance when Verifying Properties

Maximum size of SilkTest files

Options for legacy scripts

Recording a declaration for a browser page containing many child objects

Relationship between exceptions defined in 4test.inc and messages sent to the result file

Remote testing and default browser

Using a data member in the tag for a MenuItem

Using SilkTest file functions to add information to the beginning of a file

Verifying tables in client server applications

Owner-draw List Boxes and Combo Boxes

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

An owner-draw list/combo box is a list/combo box that has the owner-draw style bit set. This is distinct from a custom object that looks like a standard list/combo box, but is not.

The following procedure describes how developers can modify an application so that SilkTest can access the text of a standard list/combo box that is owner-draw and that does not have the HasStrings style bit turned on. (If the HasStrings style bit of the owner-draw list/combo box is turned on, then you do not need to make the modifications described here.) The procedure entails modifying each owner-draw list/combo box's parent window procedure so that SilkTest can query the parent about what is in the list/combo box.

To turn on the HasStrings style bit of the owner-draw list/combo box:

1. Include `owndraw.h`, supplied with SilkTest, in your source files.
2. For each owner-draw list/combo box, add a message handler for each of the messages in `owndraw.h` (`ODA_HASTEXT`, `ODA_GETITEMTEXT`, `ODA_GETITEMTEXTSIZE`). Base the message handlers on the 'switch' statement cases below. If writing in C, add code, such as that shown below, to its parent s WndProc.

```
LONG FAR PASCAL ListParentWndProc (HWND hWnd, UINT uiMsg,WPARAM wParam,
LPARAM lParam)
{
   //Use a static for the registered message number
   static UINT uiMsgGetItem Text = 0;

   LPGETITEMTEXTSTRUCT LpGetItemText;
   USHORT usItem;
   PSZ pszItemText;

   //Register the QAP_GETITEMTEXT message if it is not registered
   if (uiMsgGetItem Text == 0)
      uiMsgGetItemText = RegisterWindowMessage("QAP_GETITEMTEXT");

   switch (uiMsg)
   {
   ...
   default;
      //Process the QAP_GETITEMTEXT message
      if (uiMsg == uiMsgGetItemText)
      {
         //lParam points to a LPGETITEMTEXTSTRUCT structure
         lpGetItemText = (LPGETITEMTEXTSTRUCT) lParam;

         //Perform the requested action
         switch (lpGetItemText->Action)
         {
```

```
         case ODA_HASTEXT:
            //Tell the QAP driver if your list box contains text
            if (your list box has text)
               lpGetItemText->bSuccess = TRUE;
            else
               lpGetItemText->bSuccess = FALSE;
            break;

         case ODA_GETITEMTEXT:
            //Return the text for the requested list item
            //(lpGetItemText->itemID is the index of the item in the
            //list/combo box -- the same number passed to LB_GETITEMDATA
            (for a list box) or CB_ GETITEMDATA (for a combo box))
            usItem = UINT (lpGetItemText->itemID);
            pszItemText = <pointer to text of item[usItem]>;
            strncpy (lpGetItemText->lpstrItemText, pszItemText,
               lpGetItemText->nMaxItemText);
           lpGetItemText->lpstrItemText[lpGetItemText->nMaxItemText-1]
               = '\0';
            lpGetItemText->bSuccess = TRUE;
            break;

         case ODA_GETITEMTEXTSIZE:
            //Return the length of the requested list item
            usItem = UINT (lpGetItemText->itemID);
            lpGetItemText->nMaxItemText = <length of item[usItem]> + 1;
            lpGetItemText->bSuccess = TRUE;
            break;
         ...
         }
      }
   }
}
```

Options for legacy scripts

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

OldStartupFunction

You can change the way a script is initialized when it is opened, by setting the `OldStartupFunction` in the `[Runtime]` section of your `partner.ini` file. By setting the option to FALSE, you can have variables, windows, and other structures initialized in the order in which they are declared. When `OldStartupFunction` is set to TRUE, the default, windows are initialized before global variables. Therefore, you cannot initialize a window member from a global variable, regardless of how they were declared, but you can initialize a global variable from a window member.

AutoGuiTarget and WinvarInitCorrectly

If you have legacy scripts that run in distributed environments or on multiple platforms and that do not compile in this release, consider the two runtime options **AutoGuiTarget** and **WinvarInitCorrectly**. They deal with problems caused by the interaction of distributed testing and GUI-specific variable initialization. Set these options in the `[Runtime]` section of your `partner.ini` file.

You may find it useful to read about the GUI Targets field that appears on the **Runtime Options** dialog.

The following code produces a runtime error, because it is not possible to initialize the GUI-specific window member `abc.ghi` so that its value is correct for both the Windows 32 and Windows 9.x platforms:

Example 1

```
window abc
msw32STRING def = "1"
```

```
mws9x STRING def = "2"
msw32, msw9x STRING ghi = def
```

To avert this problem, the runtime option **AutoGuiTarget** was added. If set to TRUE, the default value, and when networking is not enabled, the GUI target is automatically set to the platform the test is running on. If it is FALSE or if the network is enabled, and if the GUI target includes both msw32 and msw9x, the code will continue to produce an error because no assumptions can be made about the GUI target.

If you are running in a distributed environment, where you might simultaneously connect to msw32 and msw9x, you must modify your code so that the variable (in the above example, `abc.ghi`) is defined for each individual platform.

In some cases, however, setting **AutoGuiTarget** to TRUE causes a problem. Consider the following code, which produces a compile-time error on Windows, because the GUI-specific window member `abc.TF` is defined only for msw32.

Example 2

```
window abc
msw32 TextField TF

main ()
BOOLEAN bmsw32 = (GetGuiType () == msw32) ? TRUE : FALSE

if (bmsw32)
abc.TF.SetText ("some value")
```

When the GUI target excludes msw32, a compile-time error will result, because the variable `abc.TF` is defined only for msw32. When the GUI target includes msw32, the reference to `abc.TF` is never executed, and so a runtime error does not occur. However, when **AutoGuiTarget** is set to TRUE, the GUI target excludes msw32, and so the code will not compile. For this reason, **WinvarInitCorrectly** was added.

When **WinvarInitCorrectly** is set to TRUE, the default, window variables are initialized to allow correct operation in a distributed environment. When **WinvarInitCorrectly** is FALSE, window variables are initialized as if the only GUI that will ever be used is the one connected to when your script starts running. Any subsequent connections to other GUIs may see window variables that are initialized incorrectly or not at all.

Summary

Here are some guidelines regarding legacy scripts:

- We recommend that you rewrite scripts running on multiple platforms so that window variables can resolve correctly. In this case, the default settings of both options are fine.
- Scripts similar to Example 2 will work well in a distributed environment. However, if the network is disabled, you will need to set **AutoGuiTarget** to FALSE.

In a nondistributed environment set **AutoGuiTarget** to TRUE in order to run scripts resembling Example 1. However, if you have legacy scripts with code similar to both examples, set both options to FALSE, thereby restoring initialization behavior supported in earlier releases.

Declaring an Object for which the Class can Vary

This functionality is available only for projects or scripts that use the Classic Agent.

This topic describes how you can declare an object for which the class can vary. For example, you may have text on an HTML page that is a link under certain conditions.

If the class of an object varies, then you need to choose a class for the window identifier, and include both classes in the tag. The class of the window identifier for the object determines which methods and properties you can call for the object. Therefore you should choose the class that includes the wider set of

methods and properties. However, keep in mind that when the object has the other class, you should only call those methods and properties that apply to that other class.

The tag of the object should be a multitag with a tag segment for each class. The class tag should be included in square brackets in each segment. You can either use the 'multitag' statement, or you can use the 'tag' statement with pipes (|) between segments.

Example

Assume that you have text on an HTML page that is a link only under certain conditions. The caption of the text is 'Inactive Text', but when it becomes a link, the caption changes to 'Active Text'.

Choose `HtmlLink` as the class of the window identifier because `HtmlLink` includes all of the `HtmlText` methods, and also includes additional methods such as `GetLocation`(). Of course, `GetLocation`() will not return a meaningful value if you call it when the object is really just `HtmlText`.

The declaration for the object would be:

```
HtmlLink TextOrLink
// recorded as 'InactiveText', since it was recorded when it
// was HtmlText
multitag "[HtmlText]InactiveText"
  "[HtmlLink]ActiveText"
```

or

```
HtmlLink TextOrLink
// recorded as 'InactiveText', since it was recorded
// when it was HtmlText
tag "[HtmlText]InactiveText|[HtmlLink]ActiveText"
```

Drag and Drop Operations

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

SilkTest supports drag-and-drop operations on Windows. Drag-and-drop operations have three distinct parts:

- Selecting an item by pressing a mouse button.
- Moving, or dragging, the item.
- Releasing the mouse button, thereby dropping the item at a target location.

The target location can be a logical location, that is, an identifiable object in a listview, treeview, or list box, or it can be a physical location specified by x, y coordinates in a window.

Five methods support drag-and-drop operations:

- `BeginDrag`
- `BeginDragAt`
- `EndDrag`
- `EndDragAt`
- `DragMouse`

`BeginDragAt` and `EndDragAt` are general methods that work for any window. They move an item to a physical target location and operate between windows of the same application or a different application. We recommend that you use care when recording drag-and-drop operations. Do the testcase setup carefully, and while recording, avoid extraneous movements.

`EndDrag` and `BeginDrag` apply only to list box, listview, and treeview controls. They move an item to a logical target location and operate between windows of the same application or a different application.

`DragMouse` combines the functionality of the begin and end drag methods. However, `DragMouse` operates only within a single window.

Example Testcases for the Find dialog

If you were testing the Find dialog, each testcase would need to:

1. Open a new document file
2. Type text into the document
3. Position the insertion point at the top of the file
4. Select Find from the Search menu
5. Select the forward (down) direction for the search
6. Make the search case sensitive

Non-datadriven testcase

```
testcase FindTest ()
TextEditor.File.New.Pick ()
DocumentWindow.Document.TypeKeys ("Test Case<HOME>")
TextEditor.Search.Find.Pick ()
Find.FindWhat.SetText ("Case")
Find.CaseSensitive.Check ()
Find.Direction.Select ("Down")
Find.FindNext.Click ()
Find.Cancel.Click ()
DocumentWindow.Document.VerifySelText (<text>)
Case
TextEditor.File.Close.Pick ()
MessageBox.No.Click ()
```

The chief disadvantage of this kind of testcase is that it tests only one out of the many possible sets of input data to the **Find** dialog. Therefore, to adequately test the **Find** dialog, you must record or hand-write a separate testcase for each possible combination of input data that needs to be tested. In even a small application, this creates a huge number of testcases, each of which must be maintained as the application changes.

Declaring an Object for which the Class can Vary

This functionality is available only for projects or scripts that use the Classic Agent.

This topic describes how you can declare an object for which the class can vary. For example, you may have text on an HTML page that is a link under certain conditions.

If the class of an object varies, then you need to choose a class for the window identifier, and include both classes in the tag. The class of the window identifier for the object determines which methods and properties you can call for the object. Therefore you should choose the class that includes the wider set of methods and properties. However, keep in mind that when the object has the other class, you should only call those methods and properties that apply to that other class.

The tag of the object should be a multitag with a tag segment for each class. The class tag should be included in square brackets in each segment. You can either use the 'multitag' statement, or you can use the 'tag' statement with pipes (|) between segments.

Example

Assume that you have text on an HTML page that is a link only under certain conditions. The caption of the text is 'Inactive Text', but when it becomes a link, the caption changes to 'Active Text'.

Choose `HtmlLink` as the class of the window identifier because `HtmlLink` includes all of the `HtmlText` methods, and also includes additional methods such as

`GetLocation`(). Of course, `GetLocation`() will not return a meaningful value if you call it when the object is really just `HtmlText`.

The declaration for the object would be:

```
HtmlLink TextOrLink
// recorded as 'InactiveText', since it was recorded when it
// was HtmlText
multitag "[HtmlText]InactiveText"
  "[HtmlLink]ActiveText"
```

or

```
HtmlLink TextOrLink
// recorded as 'InactiveText', since it was recorded
// when it was HtmlText
tag "[HtmlText]InactiveText|[HtmlLink]ActiveText"
```

When to use the Bitmap Tool

You might want to use the **Bitmap Tool** in these situations:

- To compare a baseline bitmap against a bitmap generated during testing.
- To compare two bitmaps from a failed test.

For example, suppose during your first round of testing you create a bitmap using one of SilkTest's built-in bitmap functions, `CaptureBItmap`. Assume that a second round of testing generates another bitmap, which your test script compares to the first. If the testcase fails, SilkTest raises an exception but cannot specifically identify the ways in which the two images differ. At this point, you can open the Bitmap Tool from the results file to inspect both bitmaps.

Handling Exceptions

Default Error Handling

If a testcase fails (for example, if the expected value doesn't match the actual value in a verification statement), SilkTest by default calls its built-in recovery system, which:

- Terminates the testcase.
- Logs the error in the results file.
- Restores your application to its default base state in preparation for the next testcase.

These runtime errors are called exceptions. They indicate that something did not go as expected in a script. They can be generated automatically by SilkTest, such as when a verification fails, when there is a division by zero in a script, or when an invalid function is called.

You can also generate exceptions explicitly in a script.

However, if you do not want SilkTest to transfer control to the recovery system when an exception is generated, but instead want to trap the exception and handle it yourself, use the 4Test do...except statement.

Adding to the default error handling

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

You can also use `do...except` to perform some custom error handling, then use the re-raise statement to pass control to the recovery system as usual.

Consider this situation: The Text Editor application displays a message box if a user searches for text that does not exist in the document. So you can create a data driven testcase that verifies that the message box appears and has the correct message. But suppose you want to determine if the Text Editor application is finding false matches, that is, if it is selecting text in the document before displaying the message box. That means that you want to do some testing after the exception is raised, instead of immediately passing control to the recovery system, as shown in this example:

```
testcase Negative (SEARCHINFO Data)
   STRING sMatch
   TextEditor.File.New.Pick ()
   DocumentWindow.Document.TypeKeys (Data.sText + Data.sPos)
   TextEditor.Search.Find.Pick ()
   Find.FindWhat.SetText (Data.sPattern)
   Find.CaseSensitive.SetState (Data.bCase)
   Find.Direction.Select (Data.sDirection)
   Find.FindNext.Click ()

   do
      MessageBox.Message.VerifyValue (Data.sMessage)
   except
      sMatch = DocumentWindow.Document.GetSelText ()

      if (sMatch != "")
         Print ("Found " + sMatch + " not " + Data.sPattern)
      reraise
   MessageBox.OK.Click ()

   Find.Cancel.Click ()
   TextEditor.File.Close.Pick ()
   MessageBox.No.Click ()
```

As the example shows, following the do keyword is the verification statement, and following the except keyword are the 4Test statements that handle the exception. The exception-handling statements in this example perform the following tasks: Call the `GetSelText` method to determine what text, if any, is currently selected in the document. If the return value from the `GetSelText` method is not an empty string, it means that the application found a false match. If this is the case, then print the false match and the search string to the results file. Reraise the exception to transfer control to the recovery system and terminate the testcase.

Reraise

The reraise statement raises the most recent exception again and passes control to the next exception handler (in the preceding testcase, it passes control to the built-in recovery system). We used reraise in the preceding testcase because if the exception-handling code does not explicitly reraise the exception, flow of control passes to the next statement in the testcase. So this example performs a test after an exception is raised, prints a statement to the results file if text was selected, then calls the recovery system, which terminates the testcase, logs the error, and restores the test application to its default base state.

Trapping the exception number

Each built-in exception has a name and a number (they are defined as an enumerated data type, EXCEPTION). For example, the exception generated when a verify fails is `E_VERIFY ( 13700)`, and the exception generated when there is a division by zero is `E_DIVIDE_BY_ZERO ( 11500)`.

All exceptions are defined in `4test.inc`, in the directory where you installed SilkTest.

See Exception values.

You can use the `ExceptNum` function to test for which exception has been generated and, perhaps, take different actions based on the exception. You would capture the exception in a `do...except` statement then check for the exception using `ExceptNum`.

For example, if you want to ignore the exception E_WINDOW_SIZE_ INVALID, which is generated when a window is too big for the screen, you could do something like this:

```
do
Open.Invoke ()
except
if (ExceptNum () != E_WINDOW_SIZE_INVALID)
 reraise
```

If the exception is not E_WINDOW_SIZE_INVALID, the exception is reraised (and passed to the recovery system for processing). If the exception is E_ WINDOW_SIZE_INVALID, it is ignored.

Defining your own exceptions

In addition to using built-in exceptions, you can define your own exceptions and generate them using the raise statement.

Consider the following testcase:

```
testcase raiseExample ()
   STRING sTestValue = "xxx"
   STRING sExpected = "yyy"
   TestVerification (sExpected, sTestValue)

TestVerification (STRING sExpected, STRING sTestValue)
  if (sExpected == sTestValue)
    Print ("Success!")
  else
    do
      raise 1, "{sExpected} is different than {sTestValue}"
    except
print ("Exception number is {ExceptNum()}")
      reraise
```

The TestVerification function tests two strings. If they are not the same, they raise a user-defined exception using the raise statement.

Raise Statement

The raise statement takes one required argument, which is the exception number. All built-in exceptions have negative numbers, so you should use positive numbers for your user-defined exceptions. raise can also take an optional second argument, which provides information about the exception; that information is logged in the results file by the built-in recovery system or if you call ExceptLog.

In the preceding testcase, raise is in a do...except statement, so control passes to the except clause, where the exception number is printed, then the exception is reraised and passed to the recovery system, which handles it the same way it handles built-in exceptions.

Here is the result of the testcase:

```
Testcase raiseExample - 1 error
Exception number is 1
yyy is different than xxx
Occurred in TestVerification at except.t(31)
Called from raiseExample at except.t(25)
```

Note that since the error was reraised, the testcase failed.

Using do...except statements to trap and handle exceptions

Using do...except you can handle exceptions locally, instead of passing control to SilkTest's built-in error handler (which is part of the recovery system). The statement has the following syntax:

```
do
<statements>
except
<statements>
```

If an exception is raised in the `do` clause of the statement, control is immediately passed to the except clause, instead of to the recovery system.

If no exception is raised in the do clause of the statement, control is passed to the line after the except clause. The statements in the except clause are not executed.

Consider this simple testcase:

```
testcase except1 (STRING sExpectedVal, STRING sActualVal)

do
 Verify (sExpectedVal, sActualVal)
 Print ("Verification succeeded")
except
 Print ("Verification failed")
```

This testcase uses the built-in function Verify, which generates an exception if its two arguments are not equivalent. In this testcase, if `sExpectedVal` equals `sActualVal`, no exception is raised, `Verification succeeded` is printed, and the testcase terminates. If the two values are not equal, Verify raises an exception, control immediately passes to the `except` clause (the first Print statement is not executed), and `Verification failed` is printed.

Here is the result if the two values "one" and "two" are passed to the testcase:

```
Testcase except1 ("one", "two") - Passed
Verification failed
```

Note that testcase passes and the recovery system is not called, because you handled the error yourself.

See Adding to the default error handling to learn how you can handle the error yourself and call the recovery system too.

You handle the error in the except clause. You can include any 4Test statements, so you could, for example, choose to ignore the error, write information to a separate log file, log the error in the results file, and so on. For some typical uses of `do...except`, see:

- Performing more than one verification in a testcase.
- Adding to the default error handling.

Programmatically logging an error

Testcases can pass, even though an error has occurred, because they used their own error handler and did not specify to log the error. If you want to handle errors locally and generate an error (that is, log an error in the results file), you can do any of the following:

- After you have handled the error, reraise it using the `reraise` statement and let the default recovery system handle it.
- Call any of the following functions in your script:

LogError (string, [cmd-line])	Writes string to the results file as an error (displays in red or italics, depending on platform) and increments the error counter. This function is called automatically if you don't handle the error yourself. `cmd-line` is an optional string expression that contains a command line.
LogWarning (string)	Same as `LogError`, except it logs a warning, not an error.
ExceptLog ( )	Calls `LogError` with the data from the most recent exception.

Performing more than one verification in a testcase

This functionality is available only for projects or scripts that use the SilkTest Classic Agent.

Usually, testcases have only one verification statement. If the verification fails, an exception is raised and the testcase terminates. But sometimes for ease of maintenance, you might want to have more than one verification statement in a testcase. Consider the following:

```
testcase MultiVerify ()
TextEditor.Search.Find.Pick ()
Find.VerifyCaption ("Find")
Find.VerifyFocus (Find.FindWhat)
Find.VerifyEnabled (TRUE)
Find.Cancel.Click ()
```

That testcase contains three verification statements. However, if, for example, the first verification (`VerifyCaption`) fails, an exception is raised and the testcase terminates. The second and third verifications never happen.

To verify more than one thing in a testcase, you can trap all but the last one in a `do...except` statement, such as:

```
testcase MultiVerify2 ()
   TextEditor.Search.Find.Pick ()
   do
      Find.VerifyCaption ("Find")
   except
      ExceptLog ()
   do
      Find.VerifyFocus (Find.FindWhat)
   except
      ExceptLog ()
   Find.VerifyEnabled (TRUE)
   Find.Cancel.Click ()
```

Here, all the verifications will happen each time the testcase is run. If either of the first two fails, the 4Test function `ExceptLog` is called. That function logs the error information in the results file, then continues execution of the script.

Writing an error-handling function

If you want to customize your error processing, you will probably want to write your own error-handling function, which you can reuse in many scripts.

For example, you might want to print the text associated with the exception as well as the function calls that generated the exception. The following testcase illustrates this.

```
testcase VerifyTest ()
   STRING sTestValue = "xxx"
   STRING sExpectedValue = "yyy"
   CompValues (sExpectedValue, sTestValue)
```

```
CompValues (STRING sExpectedValue, STRING sTestValue)
   do
      Verify (sExpectedValue, sTestValue)
   except
      ErrorHandler ()

ErrorHandler ()
   CALL Call
   LIST OF CALL lCall
   lCall = ExceptCalls ()
   Print (ExceptData ())
   for each Call in lCall
      Print("Module: {Call.sModule}",
      "Function: {Call.sFunction}",
      "Line: {Call.iLine}")
```

Note the following about this testcase:

- It calls the user-defined function `CompValues`, passing two arguments.
- `CompValues` uses Verify to compare its arguments. If they are not equal, an exception is automatically raised.
- If an exception is raised, `CompValues` calls a user-defined function, `ErrorHandler`, which handles the error. This is a general function that can be used throughout your scripts to process errors the way you want.
- `ErrorHandler` uses two built-in exception functions, `ExceptData` and `ExceptCalls`.

`Except Data`: All built-in exceptions have message text associated with them. `ExceptData` returns that text.

`ExceptCalls` returns a list of the function calls that generated the exception. You can see from `ErrorHandler` above, that `ExceptCalls` returns a `LIST OF CALL` (CALL is a built-in data type that is a record with three elements: `sFunction`, `sModule`, and `iLine`). `ErrorHandler` processes each of the calls and prints them in the results file.

- SilkTest also provides a function `ExceptPrint`, which combines the features of `ExceptCalls`, `ExceptData`, and `ExceptNum`.

```
Testcase VerifyTest - Passed
*** Error: Verify value failed - got "yyy", expected "xxx"
Module: Function: Verify Line: 0
Module: except.t Function: CompValues Line: 121
Module: except.t Function: VerifyTest Line: 112
```

The second line is the result of printing the information from `ExceptData`. The rest of the lines show the processing of the information from `ExceptCalls`.

And note that this testcase passes, because the error was handled locally and not reraised.

Using the Runtime Version of SilkTest

Overview of SilkTest Runtime

SilkTest Runtime provides a subset of the functionality of SilkTest. Specifically, it allows you to perform all of the SilkTest tasks associated with executing tests and analyzing results. You are prohibited from editing existing or creating new automation. SilkTest Runtime is intended to run previously compiled files. If you update a shared file while SilkTest Runtime is open, you must close SilkTest Runtime and reopen it in order to use the updated file.

SilkTest Runtime is an installation option. For additional information, refer to the *SilkTest Installation Guide*.

For a comparison of the menus and commands that are available in SilkTest with the menus and commands that are available from SilkTest Runtime, see *SilkTest Runtime Menus*.

The SilkTest Runtime Help includes the topics that are available from the full version of SilkTest, and additional product-specific information.

Installing SilkTest Runtime

SilkTest Runtime is an installation option. For additional information, refer to the *SilkTest Installation Guide*.

We strongly recommend that you do not install SilkTest Runtime on the same machine as SilkTest. SilkTest Runtime shares files with this product and will overwrite any other SilkTest installation you already have on your machine.

Note: SilkTest Runtime is sold and licensed separately from standard SilkTest.

Starting SilkTest Runtime

You can start SilkTest Runtime from the following locations:

- The command-line prompt in a DOS window by entering `runtime.exe`. The same syntax applies as with starting SilkTest from the command line.
- the SilkTest GUI if you have selected SilkTest Runtime option during installation.

When you start SilkTest Runtime, it displays minimized as an icon only; click the icon to maximize the **SilkTest Runtime** window.

Glossary

4Test compatible information or methods

Information or methods that can be passed by value in 4Test prototypes.

Abstract Windowing Toolkit

The Abstract Windowing Toolkit (AWT) is a library of Java GUI object classes that is included with the Java Development Kit from Sun Microsystems. The AWT handles common interface elements for windowing environments including Windows.

The AWT contains the following set of GUI components:

- Button
- CheckBox
- CheckBox Group (RadioList)
- Choice (PopupList)
- Label (StaticText)
- List (ListBox)
- Scroll Bar
- Text Component (TextField)
- Menu

accented character

A character that has a diacritic attached to it.

Agent

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

With the Classic Agent, one Agent process can run locally on the host machine, but in a networked environment, the host machine can connect to any number of remote Agents simultaneously or sequentially. You can record and replay tests remotely using the Classic Agent.

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

applet

A Java program designed to run inside a Java-compatible Web browser, such as Netscape Navigator.

application state

The state you expect your application to be in at the beginning of a test case. This is in addition to the conditions required for the base state. For additional information, see *Base State*.

attributes

In the test plan editor, attributes are site-specific characteristics that you can define for your test plan and assign to test descriptions and group descriptions. Each attribute has a set of values. For example, you define the Developer attribute and assign it the values of Kate, Ned, Paul, and Susan, the names of the QA engineers in your department.

Attributes are useful for grouping tests, in that you can run or report on parts of the test plan that have a given attribute value; for example, all tests that were developed by Bob can be executed as a group.

In SilkTest, an attribute is a characteristic of an application that you verify in a test case. Attributes are used in the **Verify Window** dialog box, which is available only for projects or scripts that use the Classic Agent.

Band (.NET)

Each level in the grid hierarchy has one band object created to represent it.

base state

The known, stable state you expect the application to be in at the start of each test case. For additional information, see *DefaultBaseState*.

bidirectional text

A mixture of characters that are read from left to right and characters that are read from right to left. Most Arabic and Hebrew characters, for example, are read from right to left, but numbers and quoted Western terms within Arabic or Hebrew text are read from left to right.

Bytecode

The form of Java code that the Java Virtual Machine reads. Other compiled languages use compilers to translate their code into native code, also called machine code, that runs on a particular operating system. By contrast, Java compilers translate Java programs into bytecode, an intermediate form of code that is slower than compiled code, but that can theoretically run on any hardware equipped with a Java Virtual Machine.

call stack

A call stack is a listing of all the function calls that have been called to reach the current function in the script you are debugging.

In debugging mode, a list of functions and test cases which were executing at the time at which an error occurred in a script. The functions and test cases are listed in reverse order, from the last one executed back to the first.

child object

Subordinate object in the GUI hierarchy. A child object is either logically associated with, or physically contained by, its superior object, the parent. For example, the File menu, as well as all other menus, are physically contained by the main window. For additional information, see *Parent Object*.

class

GUI object type. The class determines which methods can operate on an object. Each object in the application is an instance of a GUI class.

class library

A collection of related classes that solve specific programming problems. The Java Abstract Windowing Toolkit (AWT) and Java Foundation Class (JFC) are examples of Java class libraries.

class mapping

Association of nonstandard custom objects with standard objects understood by SilkTest.

Classic 4Test

Classic 4Test is one of the two outline editors you can use with SilkTest. Classic 4Test is similar to C and does not contain colors. Visual 4Test , enabled by default, is similar to Visual C++ and contains colors.

To switch between editor modes, click **Edit** > **Visual 4Test** to check or uncheck the check mark. You can also specify your editor mode on the **General Options** dialog box.

client area

A window's internal area, not including scroll bars, title bar, and borders.

custom object

Nonstandard object that SilkTest does not know, by default, how to interact with.

data-driven test case

Special kind of test case that receives many combinations of data from 4Test functions and/or a test plan. For additional information, see *Overview of Data-Driven Test Cases*.

data member

Variable defined within a class or window declaration. The value of a data member can be an expression, but it is important to keep in mind that data members are resolved (assigned values) during compilation. If the expression for the data member includes variables that will change at run-time, then you must use a property instead of that data member. For additional information, see *Using a Property instead of a Data Member*.

declarations

See *Window Declarations*.

DefaultBaseState

Built-in application state function that returns your application to its base state. By default, the built-in `DefaultBaseState` ensures that the application is running and is not minimized, the main window of the application is open, and all other windows, for example dialog boxes and message boxes, are closed.

diacritic

1. Any mark placed over, under, or through a Latin-based character, usually to indicate a change in phonetic value from the unmarked state.
2. A character that is attached to or overlays a preceding base character.

Most diacritics are non-spacing characters that don't increase the width of the base character.

Difference Viewer

Dual-paned display-only window that lists every expected value in a test case and its corresponding actual value. Highlights all occurrences where expected and actual values differ. You display the **Difference Viewer** by selecting the box icon in the results file.

double-byte character set (DBCS)

A double-byte character set, which is a specific type of multibyte character set, includes some characters that consist of 1 byte and some characters that consist of 2 bytes.

dynamic instantiation

This special syntax is called a dynamic instantiation and is composed of the class and tag of the object. For example, if there is not a declaration for the **Find** dialog box of the Text Editor application, the syntax required to identify the object looks like the following:

```
MainWin("Text Editor|$D:\PROGRAM FILES
   \<SilkTest install directory>\SILKTEST\TEXTEDIT.EXE").DialogBox("Find")
```

The general syntax of this kind of identifier is:

```
class("tag").class("tag"). ...
```

To create the dynamic tag, the recorder uses the multiple-tag settings that are stored in the **Record Window Declarations** dialog box. In the example shown above, the tag for the Text Editor contains its caption as well as its window ID. For additional information, see *About Tags*.

dynamic link library (DLL)

A library of reusable functions that allow code, data, and resources to be shared among programs using the module. Programs are linked to the module dynamically at runtime.

enabling

Altering program code to handle input, display, and editing of bidirectional or double-byte languages, such as Arabic and Japanese.

exception

Signal that something did not work as expected in a script. Logs the error in the results file.

frame file

See *test frame file*.

fully qualified object name

Name that uniquely identifies a GUI object. The actual format depends on whether or not a window declaration has been previously recorded for the object and its ancestors.

group description

In the test plan editor, one or more lines in an outline that describe a group of tests, not a single test. Group descriptions by default are displayed in black.

handles

A handle is an identification code provided for certain types of object so that you can pass it to a function that needs to know which object to manipulate.

Related Topics

- File handles
- HANDLE data type

hierarchy of GUI objects

Parent-child relationships between GUI objects.

host machine

A host machine is a system that runs the SilkTest software process, in which you develop, edit, compile, run, and debug 4Test scripts and test plans.

Host machines are always Windows systems.

Compare with target machine.

hotkey

If you are running Windows2000, you may need to press Alt while you are in SilkTest to toggle the hotkeys on.

The following table lists the available hotkeys and accelerator keys for each SilkTest menu:

Menu Name	Command with Hotkey	Hotkey	Accelerator Key
Breakpoint	Toggle	Alt+B+T	F5
	Add	Alt+B+A	-
	Delete	Alt+B+D	-
	Delete All	Alt+B+E	-
Debug	Abort	Alt+D+A	-
	Exit	Alt+D+X	-
	Finish Function	Alt+D+F	-
	Reset	Alt+D+E	-
	Run and Debug/Continue	Alt+D+R	F9
	Run to Cursor	Alt+D+C	Shift+F9
	Step Into	Alt+D+I	F7
	Step Over	Alt+D+S	F8
Edit	Undo	Alt+E+U	Ctrl+Z
	Redo	Alt+E+R	Ctrl+Y
	Cut	Alt+E+T	Ctrl+X
	Copy	Alt+E+C	Ctrl+C
	Paste	Alt+E+P	Ctrl+V
	Delete	Alt+E+D	Del
	Find	Alt+E+F	Ctrl+F
	Find Next	Alt+E+N	F3
	Replace	Alt+E+E	Ctrl+R
	GoTo Line	Alt+E+G	Ctrl+G
	GoTo Definition	Alt+E+O	F12
	Find Error	Alt+E+I	F4

Menu Name	Command with Hotkey	Hotkey	Accelerator Key
	Data Driven	-	-
File	New	Alt+F+N	Ctrl+N
	Open	Alt+F+O	Ctrl+O
	Save	Alt+F+S	Ctrl+S
	Save As	Alt+F+A	-
	Save All	Alt+F+L	-
	New Project	Alt+F+W	-
	Open Project	Alt+F+E	-
	Close Project	Alt+F+J	-
	Run	Alt+F+R	-
	Debug	Alt+F+D	-
	Check out	Alt+F+T	Ctrl+T
	Check in	Alt+F+K	Ctrl+K
	Print	Alt+F+P	Ctrl+P
	Printer Setup	Alt+F+I	-
	# operation file-name	Alt+F+#	Alt+F+#
	Exit	Alt+F+X	Alt+F4
Help	Help Topics	Alt+H+H	-
	Library Browser	Alt+H+L	-
	Tutorials	Alt+H+T	-
	About SilkTest	Alt+H+A	-
Include	Open	Alt+I+O	-
	Open All	Alt+I+P	-
	Close	Alt+I+C	-
	Close All	Alt+I+L	-
	Save	Alt+I+S	-
	Acquire Lock	Alt+I+A	-
	Release Lock	Alt+I+R	-
Options	General	Alt+O+G	-
	Editor Font	Alt+O+D	-
	Editor Colors	Alt+O+E	-
	Runtime	Alt+O+T	-
	Agent	Alt+O+A	-
	Extensions	Alt+O+X	-
	Recorder	Alt+O+R	-
	SilkPerformer Recorder	Alt+O+F	-

Menu Name	Command with Hotkey	Hotkey	Accelerator Key
	Class Map	Alt+O+M	-
	Property Sets	Alt+O+P	-
	SilkCentral URLs	Alt+O+U	-
	Open Options Set	Alt+O+O	-
	Save Options Set	Alt+O+S	-
	Close Options Set	Alt+O+C	-
	# option-file-name	Alt+O+#	-
Outline	Move Left	Alt+L+V	Alt+Left Arrow
	Move Right	Alt+L+R	Alt+Right Arrow
	Transpose Up	Alt+L+A	Alt+Up Arrow
	Transpose Down	Alt+L+S	Alt+Down Arrow
	Expand	Alt+L+E	Ctrl++
	Expand All	Alt+L+X	Ctrl+*
	Collapse	Alt+L+O	Ctrl+-
	Collapse All	Alt+L+L	Ctrl+/
	Comment	Alt+L+M	Alt+M
	Uncomment	Alt+L+N	Alt+N
Project	View Explorer	Alt+P+V	-
	Align	Alt+P+A	-
	&Left	Alt+P+L	-
	&Right	Alt+P+R	-
	Project Description	Alt+P+O	-
	Add File	Alt+P+D	-
	Remove File	Alt+P+R	-
Record	Window Declarations	Alt+R+W	Ctrl+W
	Application State	Alt+R+S	-
	Testcase	Alt+R+	Ctrl+E
	Method	Alt+R+T	-
	Actions	Alt+R+A	-
	Class	Alt+R+C	-
	Window Identifiers	Alt+R+I	Ctrl+I
	Window Locations	Alt+R+L	-
	Silk Performer Script	Alt+R+P	-
Results	Select	Alt+T+S	-
	Merge	Alt+T+M	-
	Delete	Alt+T+D	-

Menu Name	Command with Hotkey	Hotkey	Accelerator Key
	Extract	Alt+T+E	-
	Export	Alt+T+X	-
	Send to Issue Manager	-	-
	Convert to Plan	Alt+T+C	-
	Compact	-	-
	Show Summary	Alt+T+H	-
	Hide Summary	Alt+T+I	-
	View Options	Alt+T+V	-
	Goto Source	Alt+T+G	-
	View Differences	Alt+T+W	-
	Update Expected Value	Alt+T+U	-
	Pass/Fail Report	Alt+T+P	-
	Mark Failures in Plan	Alt+T+F	-
	Compare Two Results	Alt+T+O	-
	Next Result Difference	Alt+T+N	-
	Next Error Difference	Alt+T+r	-
Run	Compile	Alt+U+C	Alt+F9
	Compile all	-	-
	Run All Tests	Alt+U+R	F9
	Debug	Alt+U+D	Ctrl+F9
	Application State	Alt+U+A	Alt+A
	Testcase	Alt+U+T	Alt+T
	Show Status	Alt+U+S	-
	Abort	Alt+U+B	LShift+RShift
Testplan	Goto Script	Alt+T+G	-
	Detail	Alt+T+D	-
	Insert Template	Alt+T+I	-
	Completion Report	Alt+T+C	-
	Mark	Alt+T+M	-
	Mark All	Alt+T+A	-
	Unmark	Alt+T+U	-
	Unmark All	Alt+T+N	-
	Mark by Query	Alt+T+Q	-
	Mary by Named Query	Alt+T+R	-
	Find Next Mark	Alt+T+F	-
	Define Attributes	Alt+T+E	-

Menu Name	Command with Hotkey	Hotkey	Accelerator Key
	Manual tests	Alt+T+T	-
Tools	Link Tester	Alt+S+L	-
	Start SilkPerformer	Alt+S+P	-
	Data Drive Testcase	Alt+S+D	-
	Enable Extensions	Alt+S+E	-
	SilkCentral Test Manager	Alt+S+H	
View/Transcript	Expression	Alt+V+E	-
	Global Variables	Alt+V+G	-
	Local Variables	Alt+V+L	-
	Expand Data	Alt+V+X	-
	Collapse Data	Alt+V+C	-
	Module	Alt+V+M	-
	Breakpoints	Alt+V+B	-
	Call Stack	Alt+V+L	-
	Transcript	Alt+V+T	-
Window	Tile Vertically	Alt+W+T	-
	Tile Horizontally	Alt+W+H	-
	Cascade	Alt+W+C	-
	Arrange Icons	Alt+W+E	-
	Close All	Alt+W+L	-
	Next	Alt+W+N	F6
	Previous	Alt+W+P	Shift+F6
	# file-file-name	Alt+W+#	-
Workflows	Basic	Alt+K+B	-
	Data Driven	Alt+K+D	-

Hungarian notation

Naming convention in which a variable’s name begins with one or more lowercase letters indicating its data type. For example, the variable name sCommandLine indicates that the data type of the variable is STRING.

identifier

Name used in test scripts to refer to an object in the application. Logical, GUI-independent name. Identifier is mapped to the tag in a window declaration.

include file

File that contains window declarations, and can contain constant, variable, and other declarations.

internationalization or globalization

The process of developing a program core whose feature design and code design don't make assumptions based on a single language or locale and whose source code base simplifies the creation of different language editions of a program.

Java Database Connectivity (JDBC)

Java API that enables Java programs to execute SQL statements and, therefore, interact with any SQL-compliant database. Often abbreviated as JDBC.

Java Development Kit (JDK)

A free tool for building Java applets and full-scale applications. This is a "soup-to-nuts" environment which contains development and debugging tools, and documentation. Often abbreviated as JDK.

Sun and IBM both offer a JDK.

Java Foundation Classes (JFC)

Sun Microsystem's and Netscape's class library designed for building visual applications in Java. Often abbreviated as JFC.

JFC consists of a set of GUI components named Swing that adopt the native look and feel of the platforms they run on.

Java Runtime Environment (JRE)

Sun Microsystem's execution-only subset of its Java Development Kit. The Java Runtime Environment (JRE) consists of the Java Virtual Machine, Java Core Classes, and supporting files, but contains no compiler, no debugger, and no tools.

The JRE provides two virtual machines: JRE.EXE and JREW. EXE. The only difference is that JREW does not have a console window.

Java Virtual Machine (JVM)

Software that interprets Java code for a computer’s operating system. A single Java applet or application can run unmodified on any operating system that has a virtual machine, or VM.

JavaBeans

Reusable software components written in Java that perform a single function. JavaBeans can be mixed and matched to build complex applications because they can identify each other and exchange information.

JavaBeans are similar to ActiveX controls and can communicate with ActiveX controls. Unlike ActiveX, JavaBeans are platform-independent.

Often called Beans.

Latin script

The set of 26 characters (A through Z) inherited from the Roman Empire that, together with later character additions, is used to write languages throughout Africa, the Americas, parts of Asia, Europe, and Oceania. The Windows 3.1 Latin 1 character set covers Western European languages and languages that use the same alphabet. The Latin 2 character set covers Central and Eastern European languages.

layout

The order and spacing of displayed text.

levels of localization

The amount of translation and customization necessary to create different language editions. The levels, which are determined by balancing risk and return, range from translating nothing to shipping a completely translated product with customized features.

load testing

Testing that determines the actual, which means not simulated, impact of multi-machine operations on an application, the server, the network, and all related elements.

Localization

The process of adapting a program for a specific international market, which includes translating the user interface, resizing dialog boxes, customizing features if necessary, and testing results to ensure that the program still works.

localize an application

To make an application suitable for a specific locale: for example, to include foreign language strings for an international site.

logical hierarchy

The hierarchy that is implied from the visible organization of windows as they display to the user.

manual test

In the testplan editor, a manual test is a test that is documented but cannot be automated and, therefore, cannot be run within the testplan. You might chose to include manual tests in your testplan in order to centralize the testing process. To indicate that a test description is implemented manually, you use the keyword value manual in the testcase statement.

mark

In the testplan editor, a mark is a technique used to work with one or more tests as a group. A mark is denoted by a black stripe in the margin bar of the test plan. Marks are temporary and last only as long as the current work session. Tests that are marked can be run or reported on independently as a subset of the total plan.

master plan

In the testplan editor, that portion of a test plan that contains only the top few levels of group descriptions. You can expand, which means display, the sub-plans of the master plan, which contain the remaining levels of group description and test description. The master plan/sub-plan approach allows multi-user access to a test plan, while at the same time maintaining a single point of control for the entire project. A master plan file has a .pln extension. For additional information on sub-plans, see *Sub-Plan*.

message box

Dialog box that has only static text and pushbuttons. Typically, message boxes are used to prompt a user to verify an action, such as `Save changes before closing?`, or to alert a user to an error.

method

Operation, or action, to perform on a GUI object. Each class defines its own set of methods. Methods are also inherited from the class's ancestors.

minus (-) sign

In a file, an icon that indicates that all information is displayed. Click on the minus sign to hide the information. The minus sign becomes a plus sign.

modal

A dialog box that presents a task that must be completed before continuing with the application. No other part of the application can be accessed until the dialog box is closed. Often used for error messages.

modeless

A dialog box that presents a simple or ongoing task. May be left open while accessing other features of the application, for example, a search dialog box.

Multibyte Character Set (MBCS)

A mixed-width character set, in which some characters consist of more than 1 byte.

Multiple Application Domains (.NET)

The .NET Framework supports multiple application domains. A new application domain loads its own copies of the common language runtime DLLs, data structure, and memory pools. Multiple application domains can exist in one operation system process.

negative testing

Tests that deliberately introduce an error to check an application's behavior and robustness. For example, erroneous data may be entered, or attempts made to force the application to perform an operation that it should not be able to complete. Generally a message box is generated to inform the user of the problem.

nested declarations

Indented declarations that denote the hierarchical relationships of GUI objects in an application.

No-Touch (.NET)

No-Touch deployment allows Windows Forms applications, which are applications built using Windows Forms classes of the .NET Framework, to be downloaded, installed, and run directly on the machines of the user, without any alteration of the registry or shared system components.

performance testing

Testing to verify that an operation in an application performs within a specified, acceptable period of time. Alternately, testing to verify that space consumption of an application stays within specified limits.

physical hierarchy (.NET)

The window handle hierarchy as implemented by the application developer.

plus (+) sign

In a file, an icon that indicates that there is hidden information. You can show the information by clicking on the plus sign. The plus sign becomes a minus sign.

polymorphism

Different classes or objects performing the same named task, but with different execution logic.

project

SilkTest projects organize all the resources associated with a test set and present them visually in the **Project Explorer**, making it easy to see, manage, and work within your test environment.

SilkTest projects store relevant information about your project, including references to all the resources associated with a test set, such as plans, scripts, data, option sets, .ini files, results, and frame/include files, as well as configuration information, Editor settings, and data files for attributes and queries. All of this information is stored at the project level, meaning that once you add the appropriate files to your project and configure it once, you may never need to do it again. Switching among projects is easy - since you need to configure the project only once, you can simply open the project and run your tests.

properties

Characteristics, values, or information associated with an object, such as its state or current value.

For additional information, see *Using a Property Instead of a Data Member*.

query

User-selected set of characteristics that are compared to the attributes, symbols, or execution characteristics in a test plan. When the set of characteristics matches a test, the test is marked. For additional information on marks, see *Mark*. This is called marking by query. For example, you might run a query in order to mark all tests that are defined in the `find.t` script and that were created by the developer named Bob.

recovery system

A built-in, automatic mechanism to ensure the application is in a known state. If the application is not in the expected state, a message is logged to the results file and the problem is corrected. The recovery system is invoked before and after each test case is executed.

regression testing

A set of baseline tests that are run against each new build of an application to determine if the current build has regressed in quality from the previous one.

results file

In SilkTest, a file that lists information about the scripts and test cases that you ran. In the testplan editor, a results file also lists information about the test plan that you ran; the format of a results file mimics the outline format of the test plan it derives from. The name of the results file is `script-name.res` or `testplan-name.res`.

For additional information, see *Overview of the Results File*.

script

A collection of related 4Test test cases and functions that reside in a script file.

script file

A file that contains one or more related test cases. A script file has a .t extension, such as *find.t*.

Side-by-side (.NET)

Side-by-side execution is the ability to install multiple versions of code so that an application can choose which version of the common language runtime or of a component it uses.

Simplified Chinese

The Chinese alphabet used in the People's Republic of China. It consists of several thousand ideographic characters that are simplified versions of traditional Chinese characters.

Single-Byte Character Set (SBCS)

A character encoding in which each character is represented by 1 byte. Single byte character sets are mathematically limited to 256 characters.

smoke test

Tests that constitute a quick set of acceptance tests. They are often used to verify a minimum level of functionality before either accepting a new build into source control or continuing QA with more in-depth, time-consuming testing.

Standard Widget Toolkit (SWT)

The Standard Widget Toolkit (SWT) is a graphical widget toolkit for the Java platform. SWT is an alternative to the AWT and Swing Java GUI toolkits provided by Sun Microsystems. SWT was originally developed by IBM and is maintained by the Eclipse Foundation in tandem with the Eclipse IDE.

statement

In the testplan editor, lines that implement the requirements of a test plan. The testplan editor has the following statements:

- testcase
- script
- testdata
- include
- attribute

Statements consist of one of the preceding keywords followed by a colon and a value.

In SilkTest, a statement is a method or function call or 4Test flow-of control command, such as `if..then`, that is used within a 4Test test case.

status line

Area at the bottom of the window that displays the status of the current script, the line and column of the active window (if any), and the name of the script that is currently running. When the cursor is positioned over the toolbar, it displays a brief description of the item.

stress testing

Tests that exercise an application by repeating the same commands or operation a large number of times.

subplan

Test plan that is referenced by another test plan, normally the master test plan, by using an include statement. Portion of a test plan that resides in a separate file but can be expanded inline within its master plan. A subplan may contain the levels of group description and test description not covered in the master plan. A subplan can inherit information from its master plan. You add a subplan by inserting an include statement in the master plan. A subplan file has a .pln extension, as in *subplan-name.pln*.

suite

A file that names any number of 4Test test script files. Instead of running each script individually, you run the suite, which executes in turn each of your scripts and all the test cases it contains.

Swing

A set of GUI components implemented in Java that are based on the Lightweight UI Framework. Swing components include:

- Java versions of the existing Abstract Windowing Toolkit (AWT) components, such as Button, Scrollbar, and List.
- A set of high-level Java components, such as tree-view, list-box, and tabbed-pane components.

The Swing tool set lets you create a set of GUI components that automatically implements the appearance and behavior of components designed for any OS platform, but without requiring window-system-specific code.

Swing components are part of the Java Foundation Class library beginning with version 1.1.

symbols

In the testplan editor, used in a test plan to pass data to 4Test test cases. A symbol can be defined at a level in the test plan where it can be shared by a group of tests. Its values are actually assigned at either the group or test description level, depending on whether the values are shared by many tests or are unique to a single test. Similar to a 4Test identifier, except that its name begins with a $ character.

tag

This functionality is available only for projects or scripts that use the Classic Agent.

The actual name or index of the object as it is displayed in the GUI. The name by which SilkTest locates and identifies objects in the application.

target-machine

A target machine is a system (or systems) that runs the 4Test Agent, which is the software process that translates the commands in your scripts into GUI-specific commands, in essence, driving and monitoring your applications under test.

One Agent process can run locally on the host machine, but in a networked environment, the host machine can connect to any number of remote Agents simultaneously or sequentially.

Target machines can be Windows systems.

Compare with host machine.

template

A hierarchical outline in the testplan editor that you can use as a guide when creating a new test plan. Based on the window declarations in the frame file.

test description

In the testplan editor, a terminal point in an outline that specifies a test case to be executed. Test descriptions by default are displayed in blue.

test frame file

Contains all the data structures that support your scripts:

- window declarations
- user-defined classes
- utility functions
- constants
- variables

- other include files

test case

In a script file, an automated test that ideally addresses one test requirement. Specifically, a 4Test function that begins with the testcase keyword and contains a sequence of 4Test statements. It drives an application to the state to be tested, verifies that the application works as expected, and returns the application to its base state.

In a test plan, a testcase is a keyword whose value is the name of a test case defined in a script file. Used in an assignment statement to link a test description in a test plan with a 4Test test case defined in a script file.

Test case names can have a maximum of 127 characters. When you create a data driven test case, SilkTest truncates any test case name that is greater than 124 characters.

test plan

In general, a document that describes test requirements. In the testplan editor, a test plan is displayed in an easy-to-read outline format, which lists the test requirements in high-level prose descriptions. The structure can be flat or many levels deep. Indentation indicates the level of detail. A test plan also contains statements, which are keywords and values that implement the test descriptions by linking them to 4Test test cases. Large test plans can be divided into a master plan and one or more sub plans. A test plan file has a .pln extension, such as `find.pln`.

TotalMemory parameter

Total amount of memory available to the Java interpreter. This is the value returned from the `java.lang.Runtime.totalMemory()` method.

Traditional Chinese

The set of Chinese characters, used in such countries or regions as Hong Kong SAR, China Singapore, and Taiwan, that is consistent with the original form of Chinese ideographs that are several thousand years old.

variable

A named location in which you can store a piece of information. Analogous to a labeled drawer in a file cabinet.

verification statement

4Test code that checks that an application is working by comparing an actual result against an expected (baseline) result.

Visual 4Test

Visual 4Test is one of the two editors you can use with SilkTest. Visual 4Test, enabled by default, is similar to Visual C++ and contains colors. Classic 4Test is similar to C and does not contain colors.

To switch between editor modes, click **Edit** > **Visual 4Test** to check or uncheck the check mark. You can also specify your editor mode on the **General Options** dialog box.

window declarations

Descriptions of all the objects in the application's graphical user interface, such as menus and dialog boxes. Declarations are stored in an include file which has a .inc extension, typically the `frame.inc` file.

window part

Predefined identifiers for referring to parts of the window. Associated with common parts of `MoveableWin` and `Control` classes, such as `LeftEdge`, `MenuBar`, `ScrollBar`.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z