Borland
A MICRO FOCUS COMPANY

Silk Test Workbench 14.0

Getting Started with Visual Tests

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

2013-05-21

Contents

Silk Test Workbench Visual Test Tutorial

Welcome to the Silk Test Workbench Visual Test tutorial, a self-paced guide that demonstrates how to use Silk Test Workbench's visual, storyboard-based interface to create powerful and flexible functional tests. In this tutorial, you will learn the basic steps required to create a visual test, play back the visual test, and then analyze the results of the playback. Additionally, you will learn how to use a number of features that allow you to quickly update and enhance a recorded visual test.

This tutorial uses the Silk Test sample Web application, *http://demo.borland.com/InsuranceWebExtJS/*, to create a real world scenario in which you practice using Silk Test Workbench to create repeatable tests.

The lessons in this tutorial are designed to be completed in sequence as each lesson is based on the output of previous lessons.

Introducing the GUI

This section introduces the GUI including the Main Screen, Start Screen, and Visual Navigator. This section is optional. If you are already familiar with the basic elements of the development environment, proceed to the next section.

Silk Test Workbench Main Screen

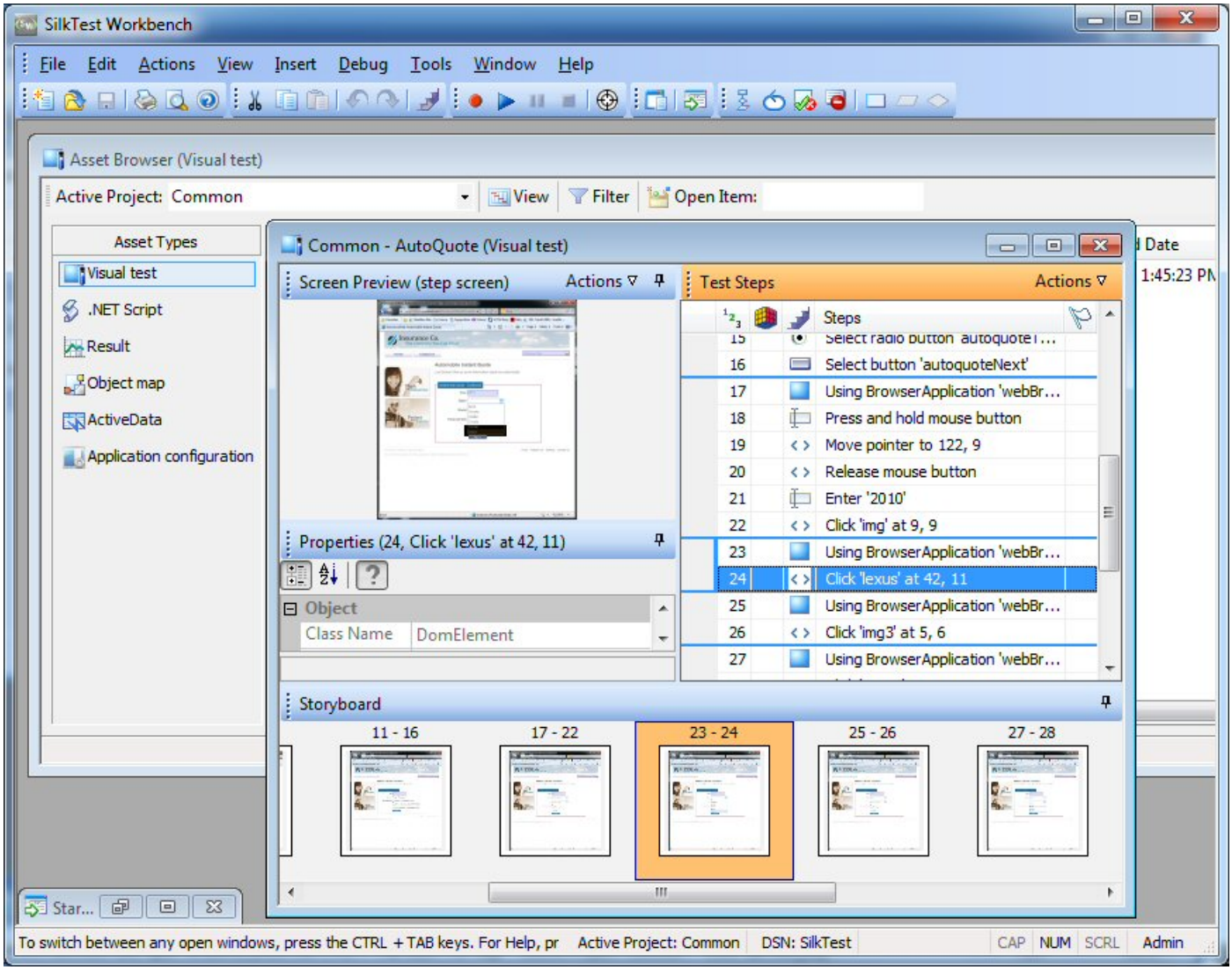
Silk Test Workbench's main screen is the starting point for all test activities. With the exception of the menu bar, the user maintains control over all of the items that are displayed on the desktop.

All other Silk Test Workbench windows, including the Start Screen, Visual Navigator, and Asset Browser, display as child windows in the Silk Test Workbench Main Screen.

The Silk Test Workbench Main Screen contains the following:

- Menu bar
- Toolbars
- Main asset viewing area
- Status bar

The following graphic shows these elements in the Silk Test Workbench Main Screen. The main asset viewing area shows a visual test in the Visual Navigator, the Asset Browser in the background, and the Start Screen, which is minimized.



Start Screen

The **Start Screen** lets you quickly begin creating test solutions for all your applications. The **Start Screen** is the launching point for creating and managing visual tests and .NET scripts, as well as links to all commonly performed testing activities. The **Start Screen** contains four panes.

Scripts Use the **Scripts** pane to access and record new visual tests and .NET scripts. You can also open, play back, or view the results of the most recently accessed visual tests and .NET scripts.

Tasks Use the **Tasks** pane to access frequently used features.

Flags Use the **Flags** pane to collaborate test suite and test project information with other testers and users of the test projects in the database.

Help The **Help** pane provides quick links to learning assistance to help get you productive quickly and lets you access product information right when you need it.

Click **Customize** to change the option to display the **Start Screen** when Silk Test Workbench starts.

Click **Close** at any time to hide the **Start Screen**.

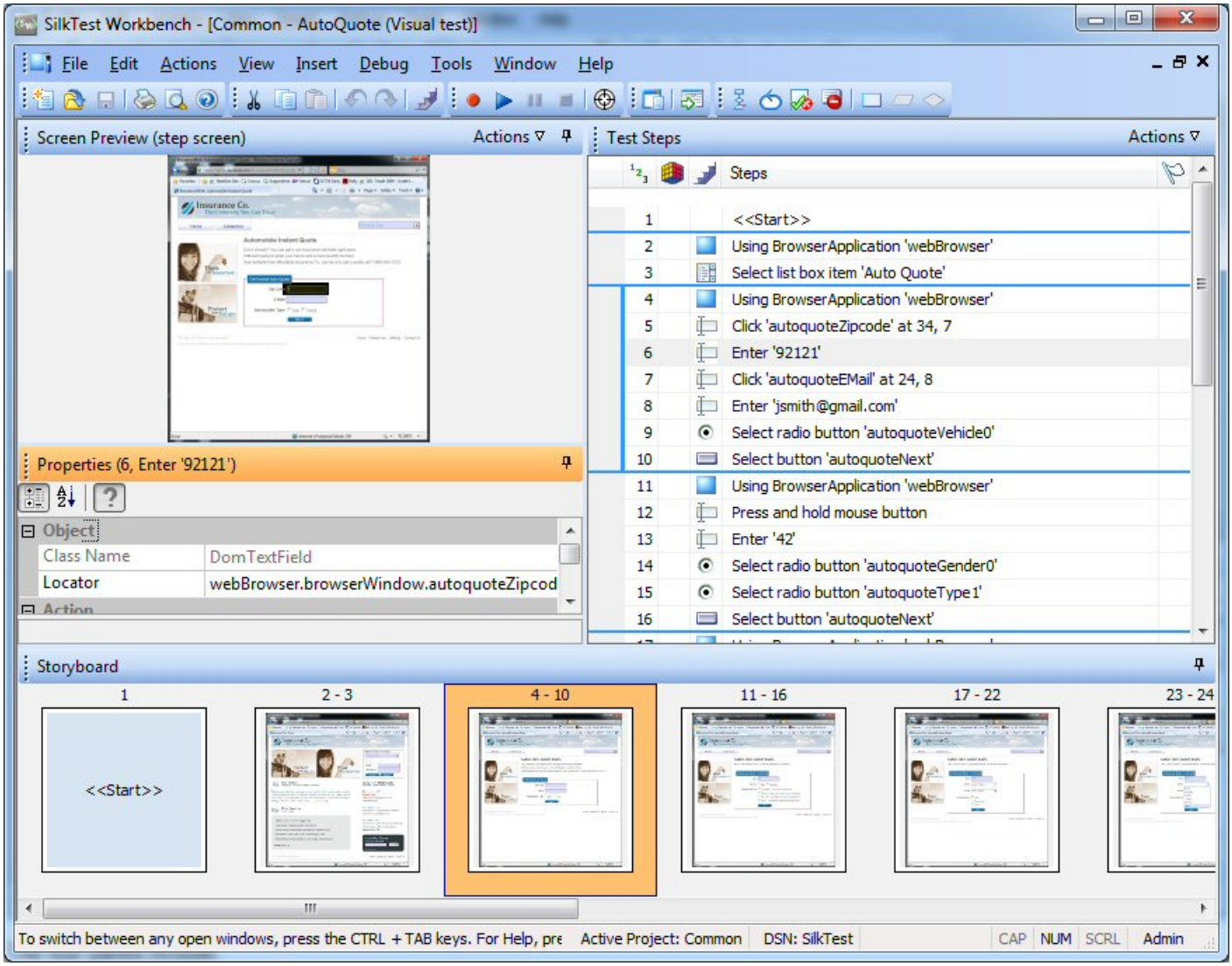
Click **Collapse Bottom Panel** to hide and **Expand Bottom Panel** to show the **Tasks** and **Help** panes in the **Start Screen**.

Tip: You can always access it from the main screen by clicking the **Start Screen** in the watermark. You can also press `Ctrl+Alt+S`, or choose **View** > **Start Screen** at any time to display it.

Note: The user interface works with standard Windows small and large font sizes. Using a custom font size may result in the inaccurate display of text in the user interface.

Visual Navigator

The **Visual Navigator** graphically represents the elements of a visual test and allows you to interact with each element through a point-and-click interface. When viewed in the **Visual Navigator**, a visual test is represented by information displayed in four panes that collectively provide a comprehensive view of each step in a visual test.



The four panes include:

Test Steps	Lists each step of a visual test in clear non-technical language.
Screen Preview	Displays a snapshot of the application under test as it appears when a step executes during playback of a visual test.

Properties	Displays the properties of a step in a visual test.
Storyboard	Displays the flow of a visual test through the use of thumbnail images, which represent the logical groups of steps in a visual test.

Note:

Silk Test Workbench takes snapshots under the following circumstances:

- Before every automation test step during recording.

  Note: For SAP applications, snapshots occur when the screen changes rather than before every automation test step.
- When executing a `Using` step in a visual test, a snapshot of the result.
- When a playback error occurs.

The **Screen Preview**, **Storyboard**, and **Properties** panes are synchronized with the **Test Steps** pane and display information specific to a selected step in the **Test Steps** pane. In this way, you can easily view all aspects of a step by selecting a step in the **Test Steps** pane, and then viewing information about the step in the other panes.

In addition to viewing a visual test, the **Visual Navigator** also allows you to enhance or update an existing visual test by using the **Screen Preview** and **Properties** panes. For example, in the **Properties** pane, after recording a visual test, you can change the literal value of a recorded property by replacing it with a variable. Additionally, to quickly update a visual test when changes occur in the application under test, you can update previously captured screens using the **Update Screen** feature of the **Screen Preview**.

The **Visual Navigator** also displays the playback result of a visual test using the same panes as those used for a visual test. For a result, the panes have additional functionality and appear in the **Result** window, which contains toolbar options and several tabs that display different views of result content. Examples of additional functionality specific to a result include the ability to see the pass or fail status of each step in the **Test Steps** pane. Additionally, in the **Screen Preview**, you can see a comparison of the differences between the screens captured during recording and screens captured during playback, and then update the existing visual test without accessing the test application.

Recording a Visual Test: Introduction

As you perform actions to create an insurance quote request in the sample Web application, Silk Test Workbench records the actions as steps. Steps form the basis of a visual test. When you have completed recording actions needed for a test, you can see the recorded test in the Visual Navigator. Steps display in the Test Steps pane of the Visual Navigator.

Note: The sample application used in this tutorial is designed and optimized to run on Internet Explorer. To ensure a user experience consistent with the lessons in the tutorial, Micro Focus does not recommend running the tutorial sample application on one of the other supported browser instead of Internet Explorer.

Note: Before you record or playback Web applications, disable all browser add-ons that are installed in your system. To disable add-ons in Internet Explorer, click **Tools** > **Internet Options**, click the **Programs** tab, click **Manage add-ons**, select an add-on and then click **Disable**.

Starting the Sample Web Application

For this tutorial, use the Silk Test sample Web application. This Web application is provided for demonstration purposes.

Use the Silk Test sample Web application with Internet Explorer. To ensure a user experience consistent with the lessons in the tutorial, we do not recommend running the sample Web application with one of the other supported browsers instead of Internet Explorer.

1. To record DOM functions to make your test faster and more reliable, perform the following steps:
   a) Click **Tools** > **Options**.
   b) Click the plus sign (+) next to **Record** in the **Options** menu tree. The **Record** options display in the right side panel.
   c) Click **xBrowser**.
   d) From the **Record native user input** list box, select **No**.
   e) Click **OK**.

   Note: Typically, when you test Web applications, you use native user input rather than DOM functions. Native user input supports plug-ins, such as Flash and Java applets, and applications that use AJAX, while high-level API recordings do not.
2. To ensure that all browser add-ons are disabled, perform the following steps:

   Before you record or playback Web applications, you must disable all browser add-ons.
   a) In Internet Explorer choose **Tools** > **Internet Options**. The **Internet Options** dialog box opens.
   b) Click the **Programs** tab and then click **Manage add-ons**. The **Manage Add-ons** dialog box opens.
   c) In the list of add-ons, review the **Status** column and ensure that the status for each add-on is **Disabled**.

      If the **Status** column shows **Enabled**, select the add-on and then click **Disable**.
   d) Click **Close** and then click **OK**.
3. To access the sample application remotely, click *http://demo.borland.com/InsuranceWebExtJS*. The sample application Web page opens.

Recording a Visual Test for the Sample Web Application

During recording, Silk Test Workbench records all interactions in the test application (except interaction with Silk Test Workbench itself) until recording is stopped. After you have finished recording, you can modify the visual test you have generated to add and remove steps.

1. Choose **File** > **New**. The **New Asset** dialog box opens.
2. Select **Visual test** from the asset types list.
3. Check the **Begin Recording** check box to start recording immediately.
4. Click **OK** to save the visual test as an asset and begin recording. The **Select Application** dialog box opens.
5. Select **InsuranceWeb: Home - Windows Internet Explorer** from the list and then click **OK**. Silk Test Workbench minimizes and a **Recording** dialog box opens.
6. In the Insurance Company Web site, perform the following steps:

   During recording the Silk Test Workbench icon on the task bar flashes. You can see the current object that you are working with and the last action that was recorded in the **Recording** dialog box.
   a) From the **Select a Service or login** list box, select **Auto Quote**. The **Automobile Instant Quote** page opens.
   b) Type a zip code and email address in the appropriate text boxes, click an automobile type, and then click **Next**.
      To follow this tutorial step-by-step, type `92121` as the zip code, `jsmith@gmail.com` as the email address and specify `Car` as the automobile type.
   c) Specify an age, click a gender and driving record type, and then click **Next**.
      For example, type `42` as the age, specify the gender as `Male` and `Good` as the driving record type.

d) Specify a year, make, and model, click the financial info type, and then click **Next**.
For example, type `2010` as the year, specify `Lexus` and `RX400` as the make and model, and `Lease` as the financial info type.
A summary of the information you specified appears.
e) Click **Purchase**.
The **Purchase A Quote** page opens.
f) Click **Home** near the top of the page to return to the home page where recording started.

7. Stop recording by pressing `Alt+F10`, clicking **Stop Recording** in the **Recording** dialog box, or clicking the Silk Test Workbench taskbar icon. The **Recording Complete** dialog box opens. If the **Do not show this message again** check box is checked in the **Recording Complete** dialog box, this dialog box does not appear after recording is stopped. In this case, the visual test displays in the **Test Steps** pane.

Saving and Naming the Visual Test

Once you have recorded your test, the **Recording Complete** dialog box opens and then you can:

- Play back the recorded visual test.
- Review the recorded actions in the Visual Navigator.
- Save the visual test, and then review the recorded test in the Visual Navigator.

Since you have just recorded the visual test for the first time, save and name the test before playing it back or reviewing the recorded actions.

1. From the **Recording Complete** dialog box, click **Save**.

   Tip: When you create an asset without naming it, Silk Test Workbench assigns the temporary name *"Untitled_"* followed by a sequential number.

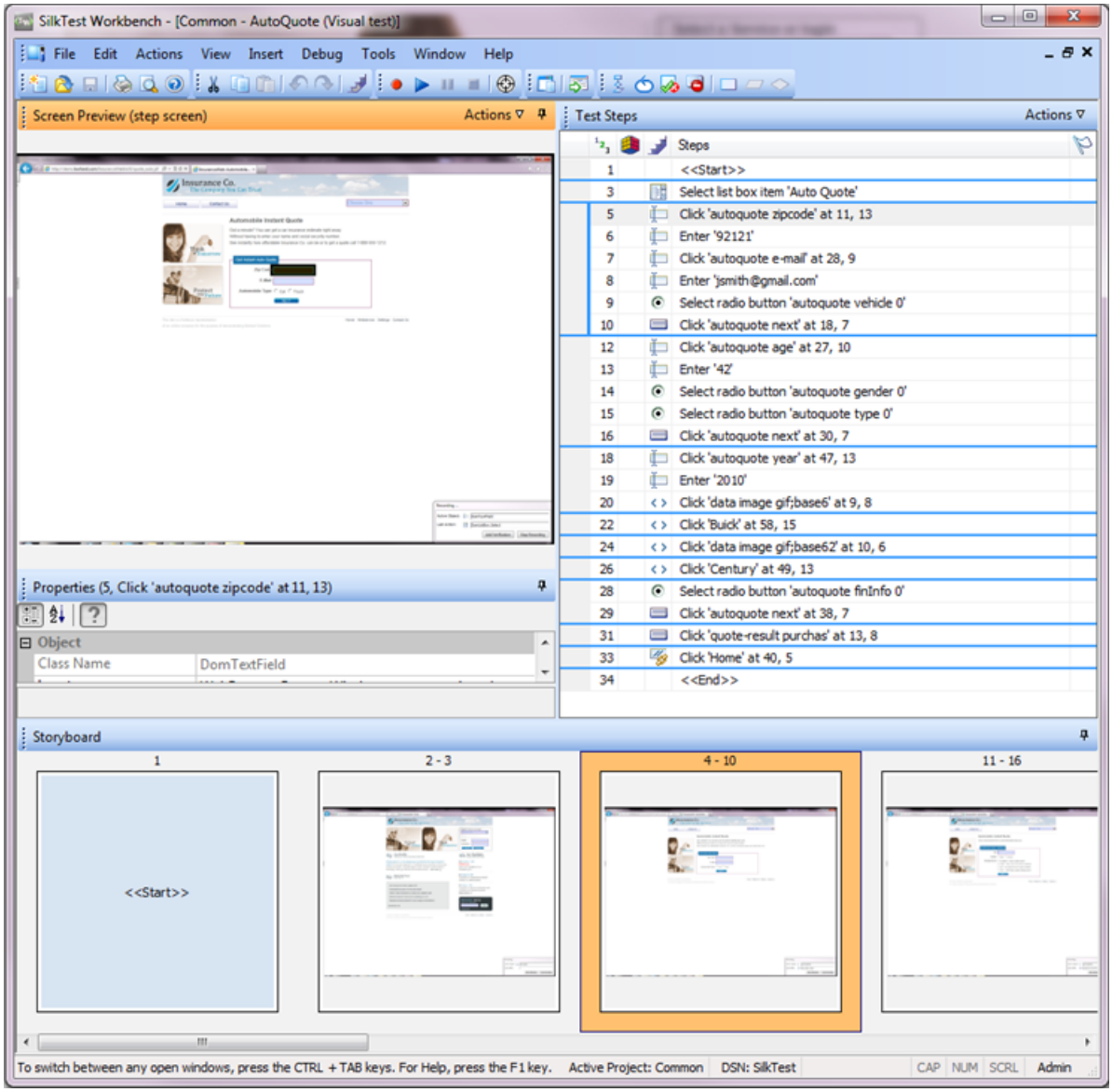
   Because the test has not been named yet, the **Save As** dialog box opens.
2. In the **Name** text box, change the name to `AutoQuote`.
3. In the **Description** text box, type `Visual test tutorial`.
4. Click **OK**. The visual test displays in the Visual Navigator.

Reviewing the Recorded Test Steps

Once you have recorded your visual test and saved it, the visual test displays in the Visual Navigator. The four panes of the Visual Navigator provide a comprehensive view of the visual test.

Steps in the Test Steps pane represent the screens accessed during recording and the actions performed while completing the quote in the Web application.

Your recorded steps should be similar to the steps in the following graphic.



The columns in the Test Steps pane include:

Column	Description
(Number)*	Represents the sequential order in which steps are recorded and played back. Steps display in this sequential order by default. Any steps that have breakpoints added for debugging display a breakpoint in the column next to this column. If you change the view from the default view (Steps and Screens) to the Screens only or the Steps only view, the numbering scheme sequentially skips the steps or screens to indicate the gap in the recording sequence.
(Logic)*	Displays icons representing the type of logic for the step, if the step contains logic.
(Step Type)*	Displays icons representing the type of action being executed in the step.
Steps	Describes the action being taken for the step.
(Flag)*	Displays the **Assigned Flag** icon.

Column	Description
	The **Assigned Flag** icon indicates that the associated step is flagged to appear in the **Test Steps** pane and optionally on the **Start Screen** of the assigned user. Additionally, you can move your pointer over the icon to display a ToolTip containing the flag description, modified date, and assigned date.
(Step Description)*	Displays a user-defined step description. This column does not display in the default view, but can be shown by clicking **Actions** from the **Test Steps** pane title bar and then **View** > **Step Description**. To create a step description, select a step. In the **Properties** pane, update the **Step description** property. A **Step Description** icon appears in this column indicating that the step has a description. Either move your pointer over the icon to display a ToolTip containing the description or select the step and read the description in the **Properties** pane.

Tip: Different panes in the Visual Navigator are synchronized with the **Test Steps** pane. In the preceding graphic, the recorded step that selects **Auto Quote** from the list box is selected in the **Test Steps** pane. As a result:

- The **Screen Preview** shows the state of the application before **Auto Quote** is selected.
- The **Properties** pane shows the properties for the selected step.
- The thumbnail representing the group of steps related to selecting the **Auto Quote** list item is highlighted in the Storyboard.

Scroll through the steps in the visual test and select various steps to view the updated information in the other panes.

Playing Back the Recorded Visual Test

Once you have recorded and saved your visual test, you can play it back to verify that the visual test works.

1. Perform one of the following steps:
   - Choose **Actions** > **Playback**.
   - Click **Playback** on the toolbar.

   The **Playback** dialog box opens. This dialog box lets you determine how the result is saved.
2. In the **Result description** text box, type `Initial test results for the recorded test`.
3. Click **OK**.

   Each result is identified with a unique test run number.

   Silk Test Workbench minimizes and the visual test plays back. During playback, the actions you performed while recording the visual test are played back on the screen against the sample application. When playback completes successfully, the **Playback Complete** dialog box opens.

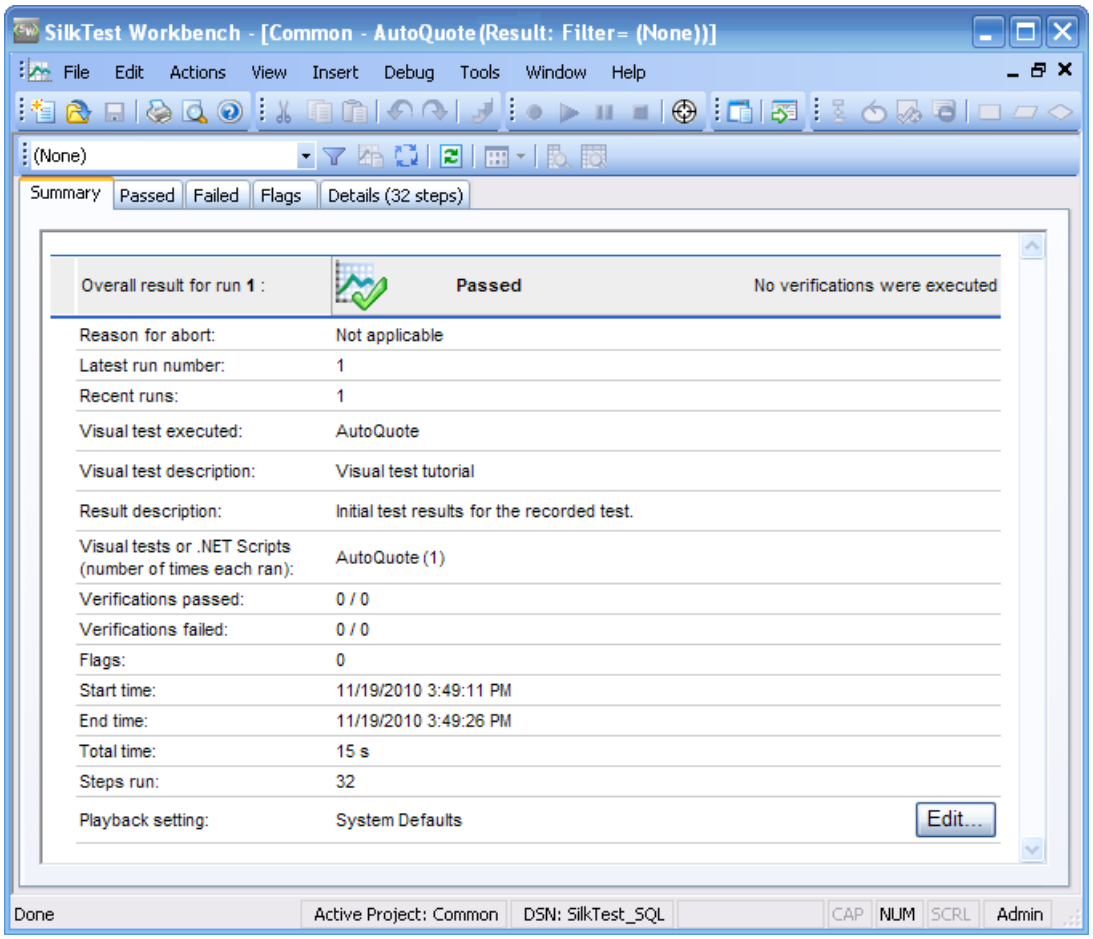
Analyzing Results: Introduction

After playing back a visual test, Silk Test Workbench generates a test result. A test result contains information about the playback of the visual test. Information such as the name of the visual test, the run number, the date and time each step executed, the pass or fail status of each step, and other important information.

Result Window

After playing back a visual test or script, you can view the results of the playback in the **Result** window. The **Result** window contains the Visual Navigator, which allows you to quickly see all aspects of test

playback. In addition to the Visual Navigator, the **Result** window contains the following additional features and functionality:



Result Window Toolbar

The **Result** window toolbar contains the following buttons and list boxes for customizing the display and type of content found in a result:

- **Filter Results By Type Selection** – Provides quick access to all pre-defined and user-defined result filters. Select a filter from the list.
- **Manage Filters** – Opens the **Manage Filters** dialog box from which you can create, edit, and apply result filters.
- **Criteria** – Displays the **Criteria** dialog box from which you can set a percentage of passed verifications as the criteria to define the success of all future runs. For example, a pass criteria of 90% means that at least 9 out of 10 verifications in a visual test or script must pass for the result of the playback to pass. Setting this option updates the Playback Result option **Result pass criteria (percentage)**. This percentage is applied to all future results.
- **Show All Runs** – Opens the **Run Detail** dialog box which displays the details of each result run. From this dialog box, you can open or delete any previous run.
- **Refresh** – Refreshes the current result.
- **View** – For visual tests, sets the type of steps to appear in the **Test Steps** pane. Click the drop-down arrow next to this button and select to view either steps only , screen steps only, or both. Additionally, you can choose to view the **Step Description** column. The selected view is applied to each tab (**Passed**, **Failed**, **Flags**, and **Details**) in the **Result** window.

- **Basic View** – Displays the standard **Test Step** pane information with the additional columns of **Result** and **Result Detail**. Disabled while viewing the **Summary** tab.
- **Advanced View** – Displays detailed information for each step. Disabled while viewing the **Summary** tab.

Result Window Tabs

The **Result** window contains five tabs that organize result content into specific types:

- **Summary tab**: Displays a high-level overview report with the following information:
  - **Overall result for run** – Indicates 'Passed' if the visual test or script played back successfully and met the result pass criteria percentage), 'Failed' if it did not meet the result pass criteria percentage, and 'Playback Error' if a step did not perform successfully.
  - **Reason for abort** – Displays the reason playback of a visual test or script was aborted.
  - **Latest run number** – Displays the run number of the most current result.
  - **Recent runs** – Displays the most recent runs. Click a previous run to view it. To open a previous run not appearing in this field, click **Show All Runs** on the toolbar to open the **Run Detail** dialog box from which you can open or delete any previous run.
  - **Visual test/.NET Script executed** – Displays the name of the visual test or script of the result.
  - **Visual test/.NET Script description** – Displays the description of the visual test or script of the result.
  - **Result description** – Displays the description of the result.
  - **Visual tests** or **.NET Scripts** – Lists all the visual tests or scripts that ran successfully as part of the playback, including inserted visual tests or scripts that ran using the `Workbench.RunScript()` method. For example, a driver script could run several scripts in one playback.
  - **Verifications passed** – The total number of verifications in all visual tests or scripts that executed successfully and passed. Click the number in this field to open **Passed** tab from which you can view all passed verifications.
  - **Verifications failed** – The total number of verifications in all visual tests or scripts that executed successfully but failed. Click the number in this field to open **Failed** tab from which you can view all failed verifications.
  - **Flags** – For visual tests, the total number of flagged verification steps in the result. Click the number in this field to open the **Flags** tab. Flags are not available for scripts.
  - **Start time** – The time the first visual test or script begins playback.
  - **End time** – The time the last visual test or script completes playback.
  - **Total time** – The total time the visual test or script played back.
  - **Steps run** – The total number of steps or code lines run.
  - **Playback setting** – Displays the Playback setting which is the group of playback options used to create the result. Click **Edit** to display the Playback options from which you can set the Playback setting.
- **Passed tab**: Displays all passed verifications.
- **Failed tab**: Displays all failed verifications. Steps that result in a playback error do not appear on this tab.
- **Flags tab**: Displays any flagged steps created by verification logic only. Flagged steps in a visual test do not appear in this tab after playback. Flags are not available for scripts.
- **Details tab**: Displays information about each step of a visual test or each code line in a script. Information such as the name of the test step or code line, the pass/fail status, a description, and flag status.

Visual Navigator Panes

For visual tests, the **Passed** tab, **Failed** tab, **Flags** tab, and **Details** tab all contain the four panes of the Visual Navigator:

- Test Steps
- Screen Preview
- Properties
- Storyboard

The **Properties** pane contains a **Show/Hide Step Properties of Visual Test Before Playback** toolbar button, which allows you to show or hide the visual test properties of a selected step.

For scripts, the **Result** window displays only the **Test Step** and **Properties** panes. The **Test Steps** pane contains additional columns which provide more information about the playback status and the result of each test step.

To customize the display of result data, you can modify the **Default result view** from the **General** options. Additionally, you can modify playback results options and create filters to view desired data.

Using the Result Window Tabs

To quickly view test result information, the **Result** window contains five tabs which function as filters that organize and display specific types of result information.

1. Click **Go to Result** on the **Playback Complete** dialog box.

   The **Result** window opens to the **Summary** tab by default. The **Summary** tab provides an overview of the test run including information such as whether the playback was successful or not, the latest run number, the number of verifications that passed or failed, the start time and end time of the test, and other basic information about the result of the test run.
2. Click the **Details** tab.

   The **Details** tab displays the result of every step using the four panes of the Visual Navigator:

Test Steps	Lists information about the playback result of each step in the visual test.
Screen Preview	Displays the Web application screens captured during playback.
Properties	Displays the properties of a step.
Storyboard	Provides a graphical outline and overview of a result.

   Note: The **Passed**, **Failed**, and **Flags** tabs also display result information using the Visual Navigator. The only difference is that these tabs display specific types of steps, whereas, the **Details** tab displays every step.
3. Select any step.

   Silk Test Workbench updates the other panes with information specific to the selected step. In the **Screen Preview**, the screen captured during playback is compared against the screen captured when the visual test was first recorded. In the **Properties** pane, the properties of the selected step are listed. And in the **Storyboard**, the group of steps in which the step occurs is highlighted.

   Tip: To view the entire name of the step, you might have to expand the **Steps** column or move your pointer over the step to display a ToolTip containing the entire name.

Using the Result Window Toolbar

The **Result** window toolbar provides several options for customizing the display and type of content found in a result.

1. Click **Advanced View** on the Results toolbar. Silk Test Workbench displays additional columns in the **Test Steps** pane and additional properties in the **Properties** pane.
2. In the **Test Steps** pane, scroll to the right to view the additional columns.

These columns provide specific information about each step such as, the time in milliseconds it took for the step to run, the user name of the person who ran the test, and other specific details.

3. In the **Properties** pane, the Result property category lists the corresponding properties of each column in the **Test Steps** pane. Scroll down to view the entire list.
4. Click the **View** drop-down arrow and select **Steps Only**. Silk Test Workbench filters out all the screens, so you can quickly see only the steps.
5. Click the **View** drop-down arrow and select **Steps and Screens**. Silk Test Workbench displays all the screens and steps of the result.

   The **Result** window toolbar contains additional options for customizing the display and type of content found in a result.

Using the Properties Pane

The **Properties** pane displays properties of a step that describe the basic characteristics of the step including the name of the step, the execution status, the details about the playback performance, and other information.

1. Click the step `Enter '42'`.

   This step performs the action of typing the value '42' into the **Age** text box.

   The **Properties** pane updates and displays the properties of the selected step.
2. Click the **Categorized** icon if it is not already selected.

   The properties are grouped into the following main categories:

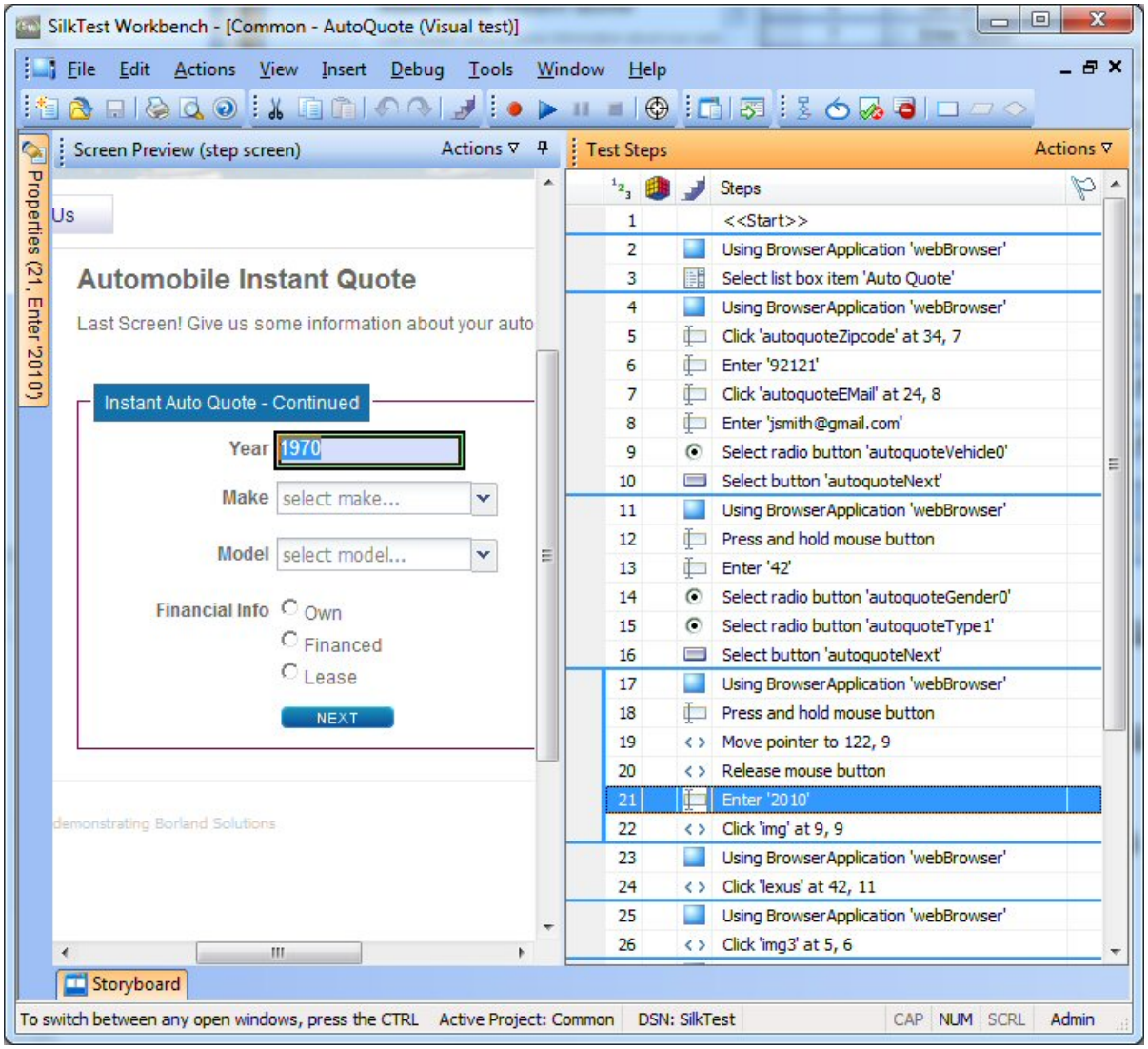
Result	Contains properties that correspond to the columns in the **Test Steps** pane. Properties such as the name of the step, the date and time the step was executed, and the time it took to run the step.
Extended properties	Contains additional playback details such as the locator name Silk Test Workbench uses to identify the control or the value of the text entered by an `Enter` action step. Extended properties are helpful to view the contents of variables or expressions when they are used in action steps. For example, an `Enter` step that uses the variable *textVar* as its value displays the contents of *textVar* in the Extended properties category.

3. Expand the **Extended properties** node. The Text property and Locator property appear with their respective values listed. The Text property value is '42', which is the value entered in the **Age** text box for this step.
4. Click the **Show/Hide Step Properties of Visual Test Before Playback** icon. The Visual test details node appears and lists all of the original properties of the step as they exist in the visual test.

Using the Screen Preview

The **Screen Preview** displays a captured image of the test application for each step in the **Test Steps** pane that interacts with a control. The captured image can show the full desktop, the application window, or only the active window. Captured images represent the state of the application before its associated step is executed.

1. In the **Test Steps** pane, select the step that contains the year of the car. Enter `2010`. In the **Screen Preview**, the screen captured during playback appears next to the screen captured during the recording of the visual test. The control for this step, the browser window, is highlighted by a black box.



2. In the **Screen Preview**, click **Actions** > **Show Differences** > **Off**. The Visual test screen closes and the Playback screen opens.
3. Click **Actions** > **Zoom** > **75%**, then use the scroll bars to position the page so that it displays the year clearly.
4. Switch to the visual test. In the **Test Steps** pane, click **Actions** > **Visual test Window**.

Enhancing the Visual Test: Introduction

Enhancing a visual test includes making updates to the existing visual test to ensure that it works with newer versions of the test application. For example, to handle and verify varying conditions in the test application you can insert test logic. Additionally, to increase the readability of a visual test or to remind yourself or others about important aspects of the test, you can insert a flag, message box, or step description.

These are just some of the ways in which you can use Silk Test Workbench to enhance existing visual test to create more powerful, robust, and flexible tests.

Updating From the Screen Preview

When Silk Test Workbench records a visual test, it captures screens from the test application in addition to the associated controls for each screen. In the **Screen Preview**, Silk Test Workbench displays each captured screen and highlights the control that is identified by a given step in the visual test. From the **Screen Preview**, you can update a step by identifying a different control in the captured screen without accessing the test application.

In this lesson, you will select a different button from the captured screen in the **Screen Preview** using Silk Test Workbench's **Insert Control From Screen Preview** feature.

1. In the **Test Steps** pane of the AutoQuote visual test, select the step before the `<<End>>` step.

   The step text is similar to the following: `Click 'Home' at 40, 5`.

   The **Screen Preview** displays the **Purchase A Quote** page and highlights the **Home** button.
2. In the **Properties** pane, click the **Locator** text box. The locator selection buttons display in the value area of the locator.
3. In the **Screen Preview**, click **Actions** > **Zoom** > **75%**, then use the scroll bars to position the screen so that the **Contact Us** button is clearly visible.
4. In the **Properties** pane, click **Identify from the screen preview**. The pointer moves to the **Screen Preview**.
5. Move the pointer over the **Contact Us** button and then click the **Contact Us** button.

   When you click the button, the **Locator** property in the **Properties** pane changes to display the new locator name for the button, and the step text in the **Test Steps** pane changes to the following: `Click 'Contact Us' at 40,5`.
6. Click **Save**.

The next time you play back the visual test, Silk Test Workbench clicks the **Contact Us** button instead of the **Home** button.

Inserting a Verification

A verification is test logic that evaluates a user-defined condition, and then sends a pass/fail message and, optionally, a flag to the playback result of a visual test.

In this lesson, you will insert a verification to ensure that the quote uses the correct vehicle model.

1. Ensure that you are viewing both steps and screens by clicking **Actions** > **View** > **Steps and Screens** in the **Test Steps** pane.
2. Select the step that reads `click 'rx400' at 91,11.`
3. In the **Screen Preview**, click **Actions** > **Zoom** > **75%**, then use the scroll bars to position the screen so that the **Model** is clearly visible.
4. Perform one of the following steps:

   - Click **Create Verification Type Logic** on the toolbar.
   - Choose **Insert** > **Test Logic** > **Verification**.

   The **Welcome** page of the **Test Logic Designer** wizard opens.
5. Click **Next**. The **Select a Logic Type** page opens.
6. Click **The property of a control**, and then click **Next**. The **Define a property-based condition** page opens.
7. Click **Identify from the screen preview**. The pointer moves to the **Screen Preview**.

8. Select the **Model** combo box. The **Define a property-based condition** page displays the name of the selected control and the properties of the control. The control name displays as:

   ```
   webBrowser.browserWindow.modelCombo
   ```

9. In the **Select a property** grid, scroll to **Text** and select it.
10. Ensure that **Select the condition** is set to **Is Equat to**.
11. Change the **Expected value** to use the updated `Text` property by typing **RX400** in the **Expected value** text box.
12. Click **Next**. The **Build the Verification** page opens. From this page, you can define the pass or fail message to send to the playback result. At the top of the page, the condition that Silk Test Workbench verifies should be as follows:

    ```
    If "webBrowser.browserWindow.modelCombo"."Text" Is Equal to "RX400"
    ```

    This condition defines the logic for the verification. The condition compares the model type to the type selected.
13. Replace the default pass text description to `The model type is correct`, and the default failed text description to `The model type is NOT correct`.
14. Click **Next**. The **Summary** page opens.

    A verification step is inserted after the selected step. The step text for the verification step is similar to the following:

    ```
    Verify "webBrowser.browserWindow.modelCombo"."Text" Is Equal to "RX400"
    ```

15. After reviewing the verification, click **Finish**.

You have successfully enhanced the recorded visual test by inserting test logic that verifies the value of a property in the sample application.

Creating a Local Variable to Store Application Data

Variables enhance visual tests by providing the ability to store data values for use in other parts of the test or in other visual tests or scripts. Data can also be output to other types of files.

In this lesson, you will store variable text in the control that displays the email address so it can be used in a later lesson of the tutorial. To do this, you must first create a local variable to store the text.

1. In the **Test Steps** pane, click **Actions** > **Insert** > **Variable** > **Add Local**. The **Add Local Variable** dialog box opens.
2. In the **Variable name** text box, type `strEmailAddress`.
3. From the **Type** list, select **Text**.

   Leave the **Initial value** text box empty for this lesson, since a value will be stored to the variable in a subsequent lesson.

   The **Text** type stores the variable value as a text, or string data type.
4. Click **OK**. The new variable is saved for the visual test. Once the variable is created, you can see it and edit its definition from the `<<Start>>` step.
5. To view the *strEmailAddress* variable, select the `<<Start>>` step in the **Test Steps** pane.

   Tip: The `<<Start>>` step is always the first step in any visual test.

   Properties for the step display in the **Properties** pane. With the `<<Start>>` step selected, *strEmailAddress* displays in the **Variables** category defined as a **Text** variable.

The value area for the variable types indicate the number of variables of the type that are currently defined. In this lesson, one Text variable has been created, so the value area for the item shows that one item is associated with this test.

Now that you have created the local variable to store the quote email address, store the email address text from the application to the variable.

Storing Application Data to the Local Variable

The sample application displays a unique email address on the **Get Instant Auto Quote page** page. The text on this page containing the email address is the property value of a control on the page.

In this lesson, you will store this text to the local variable *strEmailAddress* created in the previous exercise.

1. In the **Test Steps** pane, select the step that shows the email address value.

   The step text should look similar to the following: `Enter 'jsmith@gmail.com'`

   When the step is selected, the captured screen for the **Get Instant Auto Quote page** page displays in the **Screen Preview**.
2. In the **Test Steps** pane, click **Actions** > **Insert** > **Property from Control**.

   This inserts a step into the visual test just after the selected step. The step text should be similar to the following: `Get the '' property of the control`

   This step will be used to store the email address text from the **Get Instant Auto Quote** page to the local variable by editing the step properties. You will edit the step properties to specify the control to be used, the property of the control, and the variable to store the property value. The variable is *strEmailAddress*, which you created in the previous exercise.
3. In the **Screen Preview**, click **Actions** > **Zoom** > **50%**, and then use the scroll bars to position the screen so that the email address text is clearly visible.
4. In the **Properties** pane, click the **Locator** text box. The locator selection buttons display in the value area of the locator.
5. Click **Identify from the screen preview**. The pointer moves to the **Screen Preview**.
6. Move the pointer over the **E-Mail** text on the page.

   Ensure the highlighted box appears around the text on the page, then click the highlighted area to identify the control.

   Silk Test Workbench updates the **Locator** information in the **Properties** pane with the following value: webBrowser.browserWindow.autoquoteEMail.
7. In the **Properties** pane, click the **Property** text box, and then select **Text** from the list for its value.
8. Click **(Select a local variable...)** and then select **strEmailAddress** from the list.

   Now that you have successfully identified the control, the property of the control containing the desired value, and the variable in which to store the property value, the step text should read as follows: `Put the 'Text' property of the control into variable 'strEmailAddress'`

   The **Local variable name** value in both the **Properties** pane and the step text changes to *strEmailAddress*.

To confirm that the test captures the property value and stores it properly, play back the test and review the result.

Playing Back and Analyzing the Enhanced Visual Test

Now that you have made several enhancements to the recorded test, play back the visual test and analyze the result.

1. Perform one of the following steps:

   - Choose **Actions** > **Playback**.
   - Click **Playback** on the toolbar.

   The **Playback** dialog box opens.

2. In the **Result description** text box, type `Enhanced test results for the recorded visual test`.
3. Click **OK**. Silk Test Workbench plays back the enhanced test.
4. Click **Go to Result** on the **Playback Complete** dialog box.

   The **Result** window opens to the **Summary** tab by default.

   The **Summary** tab shows that the visual test passed, which means it played back successfully without any errors, or failed verifications.
5. Click the **Passed (1)** tab.

   The number in parentheses indicates the total number of verifications that passed. The **Test Steps** pane displays the verification step and the **Result Detail** column displays the pass text description of the verification.
6. Click the **Details** tab to display the result of every step.
7. In the **Test Steps** pane of the **Result** window, scroll down to the step that shows the result of storing the email address to a variable. The step text is similar to the following: `Put the 'Text' property of the control into variable 'strEmailAddress'`.
8. The contents of the *strEmailAddress* variable displays in the **Result Detail** column. To view the entire contents of the **Text** property value, move your cursor over the **Result Detail** column for the step. A ToolTip appears displaying the entire contents of the property.

Congratulations! You have successfully created a visual test that reliably tests the sample application. In the next lesson, you will learn about several advanced testing concepts and features such as how to quickly and easily execute a visual test within another visual test.

Executing a Visual Test Within a Visual Test: Introduction

In this tutorial, you created a single visual test that performs every action required to receive an auto insurance quote from the web application. A single visual test is useful when implementing a basic test case against a simple application. However, most software testing requires a more rigorous approach that involves testing every aspect of an application. An additional requirement is the ability to rapidly update existing visual tests whenever the test application changes.

To provide an efficient means for solving these testing challenges, Silk Test Workbench supports modular testing, in which you can "chunk" common sets of actions of a particular testing solution into a single test, and then reuse the visual test in other visual tests that require the same set of actions.

Modular Testing

Before creating visual tests, scripts, and other Silk Test Workbench assets to build application testing solutions, it is a good practice to plan a testing strategy.

It is not necessary to include all the parts of a specific test solution in a single visual test or script and is not usually beneficial to do so.

Typically, the most efficient testing approach is a modular approach. Think of your application testing in terms of distinct series of transaction units.

For example, testing an online ordering system might include the following distinct transaction units:

- Log on to the online system
- Create a customer profile
- Place orders
- Log off the online system

If one test is created to handle all of these distinct units and there are ten different scenarios that use this test, you would need to record ten different tests to handle the scenarios. If any change occurs to the

application, for example if an extra field is added to the logon window, ten different tests would require a change to accommodate data input to the new field.

Rather than creating one visual test or script that tests all of these transaction units, and then recreating it ten times for each scenario, it may be more beneficial to create separate tests as test "modules" that handle each one of these transaction units. If a separate test is created for each of the separate transaction units and reused for each of the test scenarios, then only the test that handles the logon transaction unit would require change.

Now that you understand the basics of modular testing, you are ready to create a second test and add it to the test you created in the previous lessons.

Recording the Second Visual Test

In this section of the lesson, you record a second visual test for the tutorial and learn an alternate way to create a visual test asset.

1. Choose **File** > **New**. The **New Asset** dialog box opens.
2. Select **Visual test** from the asset types list, and then type a name for the visual test asset in the **Asset name** text box.
   For this tutorial, type `AddAccount` for the name.
3. Check the **Begin Recording** check box to start recording immediately.
4. Click **OK** to save the visual test as an asset and begin recording. The **Select Application** dialog box opens.
5. Select **InsuranceWeb: Home - Windows Internet Explorer** from the list and then click **OK**. Silk Test Workbench minimizes and a **Recording** dialog box opens.
6. From the **Home** page of the sample application, click **Sign Up** in the **Login** section. The **Create A New Account** page opens.
7. Provide the following information in the appropriate fields.
   Press the `Tab` key to move from one field to the next.

Field Name	Value
First Name	Pat
Last Name	Smith
Birthday	February 12, 1990
	Note: Click the down arrow next to the month and year in the calendar control to change the month and year and then select 12 on the calendar.
E-Mail Address	smith@test.com
Mailing Address	1212 Test Way
City	San Diego
State	CA
Postal Code	92121
Password	test

8. Click **Sign Up**.
9. Click **Continue**. The contact information is displayed.
10. Click **Home** near the top of the page to return to the home page where recording started.
11. Click **Log Out**.
12. Press `Alt+F10` to complete recording. The **Recording Complete** dialog box opens.
13. Click **Save**. The visual test opens in the Visual Navigator.

Inserting One Visual Test Within Another

In this section of the lesson, you will learn how to insert the second visual test, which adds a user account, in the original visual test before the steps that perform the request for an auto quote.

Executing visual tests within visual tests is a powerful method for efficiently testing the same basic steps in multiple visual tests.

Tip: When inserting a visual test within another visual test, it is important to ensure that any test applications are in the correct initial playback state.

1. From the **Recent** list in the **Start Screen**, click `AutoQuote` to open it.

   `AutoQuote` is the first visual test that you created in this tutorial.
2. In the **Test Steps** pane, select the `<<Start>>` step.
3. Click **Actions** > **Insert** > **Visual test**. The **Browse for Visual test** dialog box opens.
4. From the **Select an asset** list, select the visual test named **AddAccount** and then click **OK**.

   Silk Test Workbench inserts a step before the selected step. The inserted step calls the selected visual test. The step text is as follows:

   ```
   Playback visual test 'AddAccount'
   ```

   Tip: During playback, when the preceding step executes, the original visual test plays back to completion before the inserted visual test plays back.

In the next lesson, you will learn how to playback this modular visual test, and respond to a playback error.

Responding to Playback Errors: Introduction

Errors encountered during playback can be caused by a variety of factors, such as changes in the test application and improper visual test step flow. Quickly diagnosing and fixing these errors using the debugging features minimizes visual test maintenance and allows for a more efficient team testing effort.

First, begin this lesson by playing back the modular test you created in the previous lesson.

Playing Back the Modular Test

In the previous lesson, you created a modular test by inserting one visual test, AddAccount, into another visual test, AutoQuote.

In this section of the lesson, you will play back the modular test and encounter an error during playback.

1. With the AutoQuote visual test open, click **Playback** on the toolbar. The **Playback** dialog box opens.
2. In the **Result description** text box, type `Responding to errors in a modular test`.
3. Click **OK**.

   During playback, the test stops on the **Create A New Account** page and an error message opens.

   This error occurs because the database requires a unique email address for each customer record. Since you have already entered the email address during the recording of the AddAccount visual test, the email address already exists in the database and the test fails.

Now that you have encountered a playback error, you are ready to debug the test.

Debugging Errors

After Silk Test Workbench encounters a playback error, the Silk Test Workbench **Playback Error** dialog box opens and provides the option to enter Debug mode. In Debug mode, playback is suspended, which allows you to diagnose and fix any playback errors using the Silk Test Workbench debugging features.

In this section of the lesson, you will learn how to debug the error that occurred during playback of the modular test in the previous section.

1. From the **Playback Error** dialog box, click **Debug**.

   In Debug mode, playback is suspended. This allows you to fix the error by either editing the properties of the step incurring the error, deleting the step, disabling the step, or copying and pasting a step from another visual test before resuming playback.

   Silk Test Workbench enters Debug mode and displays the AddAccount visual test with the step incurring the error highlighted in yellow.
2. Choose **Edit** > **Enable/Disable** to disable the step `Select button 'id=signup:continue'`.

   By disabling this step, you prevent the error from occurring the next time you play back the test.

   The step text turns grey and italicized indicating that it is disabled.
3. Click **Playback** on the toolbar. The **Playback Error** dialog box opens.
4. Click **Debug**.
5. Choose **Edit** > **Enable/Disable** to disable the step `Select button 'id=logout-form:logout'`.

   This error occurs because the test is looking for the **Log Out** button on the Home page, but since the Sign Up Continue step is disabled, the button does not display on the page. We will fix this error later in this tutorial.
6. Choose **Debug** from the menu.

   The following Step commands that appear at the top of the menu allow you to control the step execution:

Step Into (F8)	Executes playback one step at a time. **Step Into** is useful to trace through each step, and steps into other visual tests inserted into the visual test being played back. Each inserted visual test is also executed one step at a time. **Step Into** is useful for detailed analysis of a test, and lets you see the effect of each step on variable usage and test application interaction.
Step Over (`Shift + F8`)	Executes an entire visual test inserted into another visual test as if it were a single step. Use **Step Over** when playback is in debug mode for a step that plays back a visual test. This plays back the inserted visual test in its entirety. Once the inserted visual test plays back in its entirety, playback suspends in debug mode at the next step in the original visual test. Using **Step Over** at a step other than one that plays back another visual test has the same effect as using **Step Into**. Only the next step executes before playback suspends and re-enters debug mode.
Step Out (`Ctrl + Shift + F8`)	Executes all remaining steps in a visual test being played back from another visual test, then suspends playback at the next step in the original visual test. Use **Step Out** when playback is suspended at a step in a visual test that has been inserted in another visual test, and you want to playback the remaining visual test and return to the original visual test. When playback executes the remainder of the inserted visual test, it suspends and re-enters debug mode at the next sequential line in the original visual test.

Run To Cursor (`Ctrl + F8`)	Allows you to select a step where you want playback suspended. This allows you to "step over" selected sections of a visual test. Use **Run To Cursor** to playback the visual test and stop playback at a point just before a run-time error occurs. This lets you stop playback at a specific line without having to insert breakpoints. Once playback stops, you can continue using one of the other debug options.
Run From Cursor	Plays back the visual test from the currently selected test step.

7. Choose **Step Out**.

   This command executes the remaining steps in the AddAccount visual test, and then suspends playback in the next step of the AutoQuote visual test.

   After selecting **Step Out**, Silk Test Workbench displays the home page, which is the last step in AddAccount, and suspends playback. Silk Test Workbench then opens AutoQuote in Debug mode and highlights the next step.

Next, you learn how to monitor the values of variables used in the visual test.

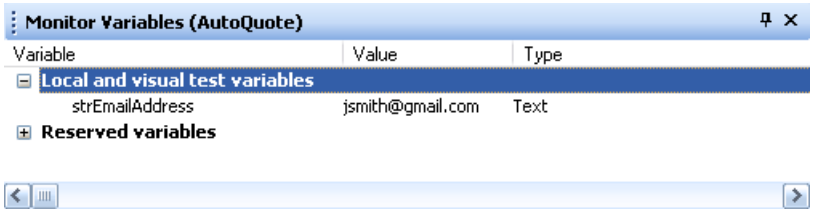
Tracking Variables During Playback

In Debug mode, you can track the playback values of both local and reserved variables using the Local Variables window.

In this section of the lesson, you will learn how to use the Local Variables window to view the value of the local variable you created in a previous lesson.

1. Choose **Debug** > **Local Variables**. The Local Variables window opens.
2. Expand the **Local and visual test variables** node.

   The local variable you created in a previous lesson, *strEmailAddress*, appears. During playback, the value for this variable will appear in the **Value** column once the step using the variable executes. The result looks like the following graphic:



Note: If you want to see the variable values in the Local Variables window but your test does not have any errors, insert a breakpoint in the visual test. To insert a breakpoint, select the step following the variable step, choose **Debug** > **Set/Clear Breakpoint** and then playback the test.

3. To see how the **Run to Cursor** debugging works, select the step `Click 'Lexus' at 24,9` and then choose **Debug** > **Run to Cursor**. Silk Test Workbench plays back the remaining steps until the step `Click 'Lexus' at 24,9`, and then displays the AutoQuote visual test.

Now you are ready to complete the playback of the modular test and review the test results.

Reviewing the Result

Review the results of the visual test after you have finished debugging the visual test.

1. Click **Playback** on the toolbar to complete playback of the AutoQuote visual test. The **Playback Complete** dialog box opens.

2. Click **Go to Result**. The AutoQuote result appears with the **Summary** tab displayed by default.

   The **Summary** tab displays the overall details of the test run. Note that the **Visual tests or .NET Scripts (number of times each ran)** field lists AutoQuote(1) and the inserted visual test, AddAccount(1).
3. Click the **Details** tab.
4. In the **Test Steps** pane, scroll down to the steps in blue text.

   By reviewing the **Result** and **Result Detail** columns, you can quickly find information about any errors that occurred during playback.

   Note: To view all steps in the **Test Steps** pane, make sure you have Steps and Screen selected. Click **Actions** > **View** > **Steps and Screens**. The **Failed** tab does not display steps containing playback errors. It only displays failed verifications.

Now that you have learned how to diagnose and debug playback errors, you are ready to modify the visual test to record additional steps to fix the error.

Modifying the Visual Test that Contains Errors

The errors within our visual test occur because the database requires a unique email address for each customer record. Since you have already entered the email address during the recording of the AddAccount visual test, the email address already exists in the database and the test fails. You can solve the database duplication error by recording additional steps within the test or you can add another modular test. For instance, we can record another visual test that resets the database and then insert the new test into the AddAccount visual test. Creating a modular test enables you to use the Reset Database steps within other visual tests as well.

1. Choose **File** > **New**. The **New Asset** dialog box opens.
2. Select **Visual test** from the asset types list, and then type a name for the visual test asset in the **Asset name** text box.
   For this tutorial, type `ResetDatabase` for the name.
3. Check the **Begin Recording** check box to start recording immediately.
4. Click **OK** to save the visual test as an asset and begin recording. The **Select Application** dialog box opens.
5. Select **InsuranceWeb: Home - Windows Internet Explorer** from the list and then click **OK**. Silk Test Workbench minimizes and a **Recording** dialog box opens.
6. From the **Home** page of the sample application, click **Settings** in the bottom right portion of the page.
7. In the **Reset Database** section, click **Reset**. This step is necessary to ensure that subsequent tests will not encounter errors when the duplicate data is entered.
8. Click **Home** near the top of the page to return to the home page where recording started.
9. Open the AddAccount visual test.
10. In the **Test Steps** pane, select the `<<Start>>` step.
11. Click **Actions** > **Insert** > **Visual test**. The **Browse for Visual test** dialog box opens.
12. From the **Select an asset** list, select the visual test named **ResetDatabase** and then click **OK**.

    Silk Test Workbench inserts a step before the selected step. The inserted step calls the selected visual test. The step text is as follows:

    ```
    Playback visual test 'ResetDatabase'
    ```
13. Open the AutoQuote visual test and click Playback on the toolbar. All three visual tests complete without any errors.

Using ActiveData

To effectively mimic application use, application testing often involves performing the same action or set of actions repeatedly using different sets of data. For example, the previous lesson included recording a test that created a customer record. To create ten customer records, you can record ten different tests, each with its own set of customer data. However, with Silk Test Workbench you can enhance this original test to run repeatedly for ten iterations and use a different set of data for each iteration.

With ActiveData testing, you can use data in external files as input to the test application, and then repeat selected steps using different data for each iteration.

In this lesson, you will learn how to:

- Create an ActiveData asset and associate it with a visual test
- Create repetition logic that repeats selected steps a specified number of times, using different data for each repetition
- Define how to use the ActiveData file in the visual test
- Define the steps to be repeated
- Map ActiveData in a data file to literal data in a visual test

In the previous lesson, you recorded a visual test that entered customer information for Pat Smith into the customer database. Each time that visual test plays back, Silk Test Workbench uses the literal data values that were captured during the initial recording.

In this lesson, you replace the literal data values used to input customer information for Pat Smith with ActiveData, so that when Silk Test Workbench plays back the visual test, different customers contained in the external file are used.

Before creating ActiveData for the visual test, review the ActiveData file.

Reviewing the ActiveData File

When creating an ActiveData asset, you can either select an existing data file or create a new file to contain the data used by the asset. In this tutorial, you will use an existing file named customers.csv, a comma-separated file containing customer information. This file is located in the Examples folder of your Silk Test Workbench installation directory.

Each column of the customers.csv file corresponds to a field used to enter a customer into the database, with the exception of the Age column, which will be used in a later lesson in this tutorial.

When using ActiveData for this test, Silk Test Workbench plays back the test and repeats the steps that enter customer information, each time using data for a different customer, until each customer in the file is entered into the database.

Tip: In the ActiveData file, password values are encoded text.

The ActiveData file contains the following columns:

- First Name
- Last Name
- Birthday
- Email Address
- Mailing Address
- City
- State
- Postal Code

- Password
- Age (used in a subsequent lesson)

Next, create the ActiveData test asset and associate it with the visual test.

Creating the ActiveData Test Asset

Before using a data file in ActiveData testing, you must create an ActiveData asset that uses the file, and then associate the ActiveData asset with the visual test.

1. In the **Recent** list on the Start Screen, double-click the visual test named **AddAccount** to open it. The AddAccount visual test opens in the Visual Navigator.
2. In the **Test Steps** pane, click **Actions** > **Insert** > **ActiveData** > **New**. The **ActiveData asset setup** window opens.
3. In the **Name** text box, name the ActiveData asset by typing `customers`.
4. Click **Browse** to search for the `customers.csv` file to associate with the AddAccount ActiveData asset. The **Choose ActiveData Asset** dialog box opens.
5. Navigate to the sample file location and select the sample ActiveData file, `customers.csv`.

   By default, the location is: `C:\Program Files\Silk\Silk Test\examples\customers.csv`.
6. Click **Open**. The path and file name appear in the **File** text box of the **General** tab.
7. In the **ActiveData asset setup** window, click the **Options** tab.
8. Check the **Use first row as header** check box.

   Tip: Click the **Details** tab to view the contents of the ActiveData file.

   This setting treats the first row of data in the ActiveData file as a header row, not as data.
9. Click **Save and close** to create the ActiveData asset and associate it with the AddAccount visual test.

In a visual test, information about any associated ActiveData assets is stored in the **Properties** pane of the `<<Start>>` step. To review each ActiveData asset associated with the visual test , select the `<<Start>>` step and review the **ActiveData** property in the **Properties** pane.

Now that the ActiveData asset has been created and associated with the visual test, you are ready to create the repetition logic that will use the active data in the test.

Creating Repetition Logic for ActiveData Files

In visual tests, ActiveData typically involves repeating a series of steps, and substituting literal data for ActiveData for each repetition.

In this lesson, we will substitute the literal data that adds a customer to the database with data from the ActiveData file customers.csv, which contains ten customer records. For each repetition, data from a customer record in the ActiveData file will be used to enter data in the fields on the **Create a New Account** page of the InsuranceCo Web site.

1. Make sure the AddAccount visual test displays in the Visual Navigator.
2. Click **Create Repetition Type Logic** on the Silk Test Workbench toolbar. The **Test Logic Designer** wizard appears with the **Welcome** page displayed.
3. Click **Next**. The **Select a Logic Type** page opens.
4. Click **Repeating a sequence of steps using data from an ActiveData file**, and then click **Next**. The **Define the ActiveData asset to use** page opens.
5. Since only one ActiveData asset (customers) has been associated with the visual test, it should appear in the **ActiveData asset** list. If not, select it from the list.
6. In the **Start row** text box, leave the default value at **1**.

7. Check the **End at last row containing data** check box.
8. Click **Get all rows in sequence**.

   You have now defined how the data file will be used in the repetition logic, and how many repetitions will occur.

   - The first row of the Active data file will be used first when the repetition logic executes.
   - The repeat will continue using each row until the last row is used.
   - Since all rows of data will be used in the repetition logic in sequential order and there are 10 rows of data in the ActiveData file, the steps to add a customer to the database will be repeated 10 times. For each repetition, a different row of customer data from the ActiveData file will be used as input.
9. Click **Next**. The **Build the Repeat** page opens.

Now that you have defined how the ActiveData will be used, you are ready to determine the steps in the visual test that will be repeated.

Defining the Steps to Repeat

When determining the steps in repetition logic to repeat, consider all the actions that require repetition, not just the steps where literal data is substituted.

For this lesson, there are several steps where literal data will be substituted, but the entire process of accessing the **Create a New Account** page of the InsuranceCo Web site must be repeated in order for each customer in the ActiveData file to be entered. Therefore, all steps in the actual test process must also be repeated, including steps that access the page and return the test to the **Home** page.

1. In the **Build the Repeat** page, click the **Do step** link. The **Select Steps** dialog box opens.
2. Select the step that begins interaction with the InsuranceCo Web site. This is the step immediately after the `<<Start>>` step.

   Because the visual test contains all the steps required in the process of adding a customer, all the steps must be repeated.
3. Click **OK**. The **Select Steps** dialog box closes and the text for the selected step appears in the **Do step** link.
4. In the **Build the Repeat** page, click the **To step** link. The **Select Steps** dialog box opens.
5. Select the step that ends interaction with the InsuranceCo Web site. This is the step immediately before the `<<End>>` step.
6. Click **OK**. The **Select Steps** dialog box closes and the text for the selected step appears in the **To step** link.
7. Click **Next**. The **Summary** page of the **Test Logic Designer** opens. This page displays the test logic you have defined for the visual test.
8. Click **Finish** to insert the test logic into the test steps of your visual test.

   In the visual test, Silk Test Workbench inserts a step after the `<<Start>>` step that starts the repetition logic. The step text should be as follows:`Repeat using activedata 'customers'`

   Silk Test Workbench also inserts a step before the `<<End>>` step that ends the repetition logic. The step text should be as follows:

   ```
   End Repeat
   ```

   All steps between the newly inserted repetition logic steps now appear as indented steps nested between the repetition logic steps. The final part of using ActiveData in the visual test is to map data in the ActiveData file to the literal data in the visual test.

Mapping ActiveData to Literal Data

Before a visual test can use data in an ActiveData file, data in the applicable test steps must be mapped to use data in columns of an ActiveData file.

1. Select the first step in the visual test that enters data into a field.

   This should be the step that enters 'Pat' into the **First Name** text box on the **Create a New Account** page of the InsuranceCo Web site, as originally recorded in the visual test.

   Properties for the step appear in the **Properties** pane.
2. In the **Properties** pane, in the **Parameters** category, select the **text** parameter, which indicates the text that was set to **Pat**.
3. Click **Select** in the value area and click **ActiveData**.

   Tip: If **customers** is not in the list **ActiveData asset** list, it has not been associated with the visual test.

   The **Select a Property, Variable, ActiveData or Edit a Literal** dialog box opens. Use this dialog box to map the data from the selected property in the test step to data from a column in the ActiveData file.

   The **ActiveData asset** text box shows the ActiveData asset **customers**, which was associated with the visual test. The column names from the customers ActiveData file display in the **Columns** list.
4. In the **Columns** list, select the **First Name** column.

   This is the column in the file that contains first names, which you are mapping to substitute for the literal data 'Pat'.
5. Click **OK** to close the dialog box and map the data. Silk Test Workbench replaces the actual data in the test step with an expression that maps to the selected data in the ActiveData file. The existing step text:

   ```
   Enter 'Pat'
   ```

   now shows the name of the ActiveData asset, the type of data being substituted, and the column name:

   ```
   Enter '[[customers].Text("First Name")]'
   ```
6. Repeat steps 1 through 5 for all other test steps containing data to be substituted with data in an ActiveData file associated with the visual test. Start with the next step, which is the step containing the step text `Enter 'Smith'` text.

   The following table shows:

   - The text of each step in this visual test whose literal date is to be substituted with data from the ActiveData file
   - The name of the step property whose data is to be substituted
   - The ActiveData column to select that contains the data that will be used in place of the literal data

   Note: Steps updated using ActiveData that enter text into a password protected text box do not show ActiveData information in the step text.

Step Text	Property	ActiveData Column
Enter 'Smith'	Text	Last Name
Enter 'smith@test.com'	Text	Email Address
Enter '1212 Test Way'	Text	Mailing Address
Enter 'San Diego'	Text	City
Select list box item 'California'	List Box Item	State
Enter '92121'	Text	Postal Code
Enter 'test'	Text	Password

Now that this visual test has been set up to use data from an ActiveData file, play it back and review the results.

Playing Back and Analyzing the ActiveData Visual Test

Now that the visual test that adds a customer has been enhanced to use customer records from an ActiveData file, it can be played back. When the visual test plays back, the customer records from the ActiveData file will actually be entered into the customer database.

1. Perform one of the following steps:
   - Choose **Actions** > **Playback**.
   - Click **Playback** on the toolbar.

   The **Playback** dialog box opens.
2. In the **Result description** text box, type `AddCustomers using ActiveData`.
3. Click **OK**. Silk Test Workbench plays back the enhanced test.
4. Click **Go to Result** on the **Playback Complete** dialog box.

   The **Result** window opens to the **Summary** tab by default.
5. Click the **Details** tab to display the result of every step.

   The **Details** tab displays the result of every step executed during playback. Use the scrollbar to scroll though the executed steps.

   There are more steps in the **Test Steps** pane of the **Result** window than there are actual steps in the visual test. This is because each repetition includes its own set of executed steps.

Note: You can also use expressions to update the ActiveData in a visual test. The **Expression Designer** enables you to use data from an ActiveData file column as input, and create an expression that changes the value of the data. The updated data can then be saved back to the same column in the same ActiveData file. For details, see *Updating ActiveData in a Visual Test Using an Expression* in the online help.

Playing Back Scripts From Visual Tests

Scripts and visual tests are similar in that both assets automate manual user actions such as selecting menu items and entering data in a test application. The difference between these two assets is how the user actions are represented in the asset. A script uses a scripting language, Microsoft's Visual Basic running in the Microsoft .NET framework, whereas, a visual test uses steps generated by the point-and-click interface of the Visual Navigator.

With either asset you can create powerful and flexible automated tests that run independently of each other. The choice of asset is dependent on your needs and preferences. You can also use each asset in conjunction with the other. For example, you can create a script that performs a specialized task, and then insert a step in a visual test that plays back the script. In this way, you can leverage the power of the scripting language used in scripts to supplement your visual tests.

When scripts are played back from a visual test, they are the equivalent of a function that you can call whenever you need to perform a repetitive and specialized testing task. This approach is helpful in a team testing environment, where an experienced tester can create a library of scripts that perform common testing functions from which a novice developer can select from when creating a visual test.

Creating a Script to Generate Random Numbers

The first step is to create a script that generates a random number that you can use for the age of the user.

1. Choose **File** > **New**. The **New Asset** dialog box opens.
2. Select **.NET Script** from the asset types list, type `function_randomAge` in the **Asset name** text box and click **OK**.

Tip: By naming your script "function" you can create an organized library of commonly used testing tasks from which you can quickly select and insert into a visual test.

The script opens in the **Code** window.

3. Place your cursor on the line following `Sub Main()`, and type:

```
Dim rand As New Random()
Dim TSrandomAge As Integer = rand.Next(10000, 99999)
MsgBox(TSrandomAge)
```

4. Click **Playback** on the toolbar. The **Playback** dialog box opens.
5. Click **OK**. A message box opens and displays a randomly generated number between 10,000 and 99,999.
6. Click **OK**. The **Playback Complete** dialog box opens.
7. Click **Go to .NET Script**.
8. Replace the numeric range parameters (10000, 99999) with the parameters *MinVal* and *MaxVal*. You can use these two parameters to set the range of the random number from the visual test that plays back this script instead of having to open the script and change the values. Type the following:

```
Dim TSrandomAge As Integer = rand.Next(MinVal, MaxVal)
```

9. To verify the script works properly, assign the parameters the following values after `Sub Main()`:

```
Dim MinVal=16
Dim MaxVal=105
```

Your script should look like the following code:

```
Public Sub Main()
  Dim rand As New Random()
  Dim MinVal=16
  Dim MaxVal=105
  Dim TSrandomAge As Integer = rand.Next(MinVal, MaxVal)

  MsgBox(TSrandomAge)
End Sub
```

10. Click **Playback** on the toolbar. The **Playback** dialog box opens.
11. Click **OK**. A message box appears and displays a randomly generated number between 16 and 105.
12. Click **OK**. The **Playback Complete** dialog box opens.
13. Click **Go to .NET Script**.
14. To set the *MinVal* and *MaxVal* parameters using input parameters, perform the following steps:
    a) Modify the `Main()` sub to include the input parameters that you want to create.

```
Public Sub Main(args As IDictionary(Of String, Object))
  Dim MinVal=args ("MinVal")
  Dim MaxVal=args ("MaxVal")
```

    where `"MinVal"` and `"MaxVal"` are the names of the input parameters that we will create in the next procedure.
    b) Modify the `Main()` sub to include the output parameters that you want to create.

```
Public Sub Main(args As IDictionary(Of String, Object))

  args ("TSrandomAge")=TSrandomAge
```

    where `"TSrandomAge"` is the names of the output parameters that we will create in the next procedure.
    c) Since the script will pass the age to the visual test, comment out the message box code by placing an apostrophe in front of the statement.

```
'MsgBox(TSrandomAge)
```

The entire script should look like the following:

```
Public Module Main
    Dim _desktop As Desktop = Agent.Desktop
```

```
    Public Sub Main(args As IDictionary(Of String, Object))

        Dim rand As New Random()

        Dim MinVal= args("MinVal")
        Dim MaxVal= args("MaxVal")

        Dim TSrandomAge As Integer = rand.Next(MinVal, MaxVal)
        args ("TSrandomAge")=TSrandomAge
        'MsgBox(TSrandomAge)

    End Sub
End Module
```

Next, define the script input and output parameters used to pass the random age to the visual test.

Defining the Script Input Parameters

Scripts can receive data from a visual test in an input parameter, and, conversely, pass data to visual tests in an output parameter. In this task, you will define two input parameters that set the range for the random number created in the function_randomAge script.

1. In the **Properties** pane, right-click and choose **Add Input Parameter**. The **Add Script Input Parameter** dialog box opens.
2. In the **Name** text box, type **MinVal**.
3. From the **Type** list, select **Number (Double)**.
4. In the **Default Value** text box, type `16`.

   Optionally, you can set the default value in the visual test.
5. Click **OK**. The input parameter displays in the list of input parameters in the **Properties** pane.
6. In the **Properties** pane, right-click and choose **Add Input Parameter**. The **Add Script Input Parameter** dialog box opens.
7. In the **Name** text box, type **MaxVal**.
8. From the **Type** list, select **Number (Double)**.
9. In the **Default Value** text box, type `105`.

   Optionally, you can set the default value in the visual test.
10. Click **OK**. The input parameter displays in the list of input parameters in the **Properties** pane.

Defining the Script Output Parameters

Scripts can receive data from a visual test in an input parameter, and, conversely, pass data to visual tests in an output parameter. In this task, you will define a parameter that passes the random number created in the function_randomAge script to the visual test.

1. In the **Properties** pane, right-click and choose **Add Output Parameter**. The **Add Script Output Parameter** dialog box opens.
2. In the **Name** text box, type **TSrandomAge**.
3. From the **Type** list, select **Number (Double)**.
4. Leave the **Default Value** text box empty.
5. Click **OK**. The output parameter displays in the list of output parameters in the **Properties** pane.

Define a local variable in the visual test to receive the output parameter.

Setting Up the Visual Test to Use Script Data

To set up a visual test to use script data, you must insert a step that plays back the script, create a local variable to store the script data, and associate the local variable with the script output variable. Additionally, to pass data from a visual test to a script, you must set the values of the script input variables.

1. Open the AutoQuote visual test.
2. Choose **File** > **Save As**. The **Save As** dialog box opens.
3. Since you will be modifying the AutoQuote visual test, rename it to **AutoQuote_Modified** to differentiate it from the original test, and then click **OK**.
4. Select the step after the following step text:
   `Playback visual test 'AddAccount'`
5. In the **Test Steps** pane, click **Actions** > **Insert** > **.NET Script**. The **Browse for .NET Script** dialog box opens.
6. Select **function_randomAge**, and then click **OK**. Silk Test Workbench inserts a step that plays back the script.
7. To add a local variable to store the script data, perform the following steps:
   a) In the **Test Steps** pane, click **Actions** > **Insert** > **Variable** > **Add Local**. The **Add Local Variable** dialog box opens.
   b) In the **Variable name** text box, type `VTrandomAge`.
   c) From the **Type** list, select **Number (Double)** and leave the **Initial value** set to 0.
   d) Click **OK**. You can see the variable and edit its definition from the `<<Start>>` step.
8. Select the step for the script.
   The step looks like:`Playback .NET Script 'function_randomAge'`.
9. In the **Properties** pane of the inserted script step, click in the value area of the **Pass contents of 'TSrandomAge' into** property and select the local variable **VTrandomAge**.
   Since we added input parameters in the script, those values display in the **Properties** pane. However, you can modify the input parameters used in the visual test by editing the values. These values are not changed in the script.

Your visual test is now set up to use script data. Each time the AutoQuote_Modified visual test plays back, the function_randomAge script is played back and passes a unique random number to the local variable, VTrandomAge, in the visual test.

Now, modify the step that enters the age information to use the visual test local parameter containing the random number generated in the function_randomAge script.

Using Script Data in the Visual Test

In this section of the lesson, you will modify the step that enters the age information to use the visual test local parameter containing the random number generated in the function_randomAge script.

1. Select the step that enters data in the **Age** text box. The step text is as follows:
   `Enter '42'`
2. In the **Properties** pane, select the **Text** property.
3. Click the **Select** button, and then select **Variable**. The **Select a Property, Variable, ActiveData or Edit a Literal** dialog box opens.
4. Select **VTrandomAge**, and then click **OK**. The step text displays as:`Enter '[VTrandomAge]'`

This step will now enter the value of the *VTrandomAge* local variable in the **Age** text box.

Playing Back and Reviewing Test Results

1. Perform one of the following steps:
   - Choose **Actions** > **Playback**.
   - Click **Playback** on the toolbar.

   The **Playback** dialog box opens.
2. In the **Result description** text box, type `Adding Customers using data passed from a script`.
3. Click **OK**. Silk Test Workbench plays back the enhanced test and creates 10 new customers using the random number passed from the script and the additional customer information you defined in the visual test.
4. Click **Go to Result** on the **Playback Complete** dialog box.

   The **Result** window opens to the **Summary** tab by default.
5. Click the **Details** tab to display the result of every step.

Index

V