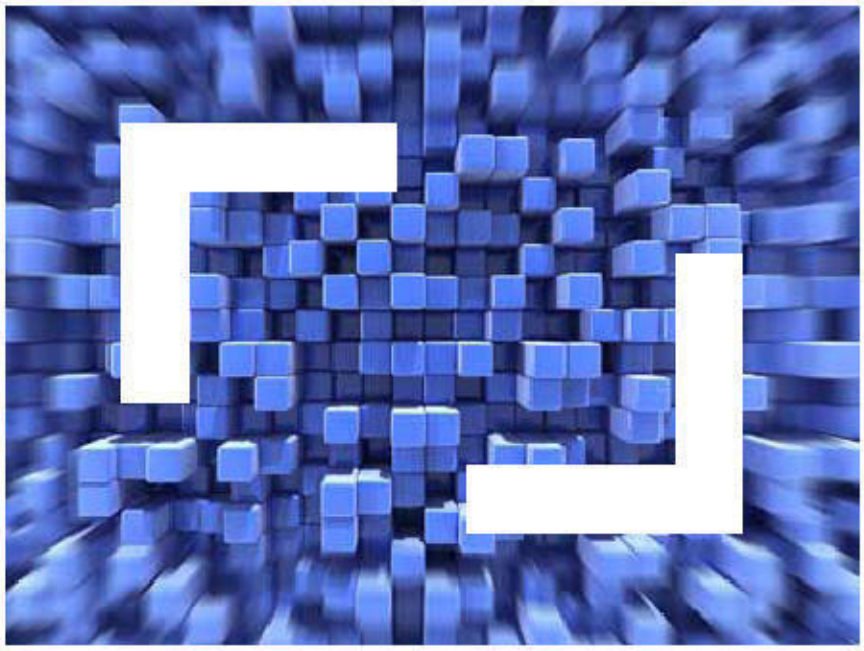


SilkTest® Workbench

Getting Started with .NET Scripts

Borland®
(A MICRO FOCUS COMPANY)

MICRO FOCUS®
Leading the Evolution™

Borland Software Corporation
4 Hutton Centre Dr., Suite 900
Santa Ana, CA 92707

Contents

Welcome to the SilkTest Workbench Script Tutorial

Welcome to the SilkTest Workbench Script tutorial. In this tutorial, you will learn the basic steps required to create a script, play back the script, 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 script.

This tutorial assumes some basic knowledge of Microsoft Visual Basic and the Microsoft .NET framework. If you are unfamiliar with the .NET framework, refer to the Microsoft web site for additional help.

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

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 the Mozilla Firefox browser.

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, choose **Tools ➤ Internet Options**, click the **Programs** tab, click **Manage add-ons**, select an add-on and then click **Disable**.

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

Recording a Script: Introduction

As you perform actions to create an insurance quote request in the sample Web application, SilkTest Workbench records the actions. When you have completed recording the actions needed for a script, you can see the recorded script in the **Code** window.

Starting the Sample Web Application

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

Use the SilkTest 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 the Mozilla Firefox browser.

1. To record DOM functions to make your test faster and more reliable, perform the following steps:
   a) Choose **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.
4. If you installed the Web application locally, perform the following steps:

   a) Choose **Start ➤ Programs ➤ Borland ➤ Borland Demo Application ➤ Startup Demo Environment**.
   The database starts.
   b) Choose **Start ➤ Programs ➤ Borland ➤ Borland Demo Application ➤ Demo Application**.
   The sample application Web page opens.

   To install the Web application locally, choose**Start ➤ Programs ➤ Silk ➤ SilkTest <version> ➤ Sample Applications ➤ xBrowser ➤ Test Application** and then extract and run the EXE file.

Recording a Script for the Sample Web Application

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

1. Choose **File ➤ New**.
The **New Asset** dialog box opens.
2. Select **.NET Script** from the asset types list, and then type a name for the script in the **Asset name** text box.
For example, type **`AutoQuote`** as the title.
3. Check the **Begin Recording** check box to start recording immediately.
4. Click **OK** to save the script 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**.
SilkTest Workbench minimizes and a **Recording** dialog box opens.
6. In the Insurance Company Web site, perform the following steps:

   During recording the SilkTest 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 SilkTest 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 script displays.
8. Click **Go to .NET Script**.
The script displays in the **Code** window.
9. Click **Save**.

Reviewing the Recorded Script

SilkTest Workbench records all actions in all applications other than itself. If you followed the instructions carefully, SilkTest Workbench captured only the actions performed on the sample application Web site. SilkTest Workbench repeats these actions during playback.

Your script should look similar to the following sample.

Note: This tutorial uses locators rather than object map items. As a result, your code may look slightly different from the tutorial.

```
Imports SilkTest.Ntf.XBrowser
Public Module Main
  Dim _desktop As Desktop = Agent.Desktop

  Public Sub Main()
    With _desktop.BrowserApplication()
    With .BrowserWindow()
      .DomListBox("@id='quick-link:jump-menu'").Select("Auto Quote")
      .DomTextField("@id='autoquote:zipcode'").SetText("92121")
      .DomTextField("@id='autoquote:e-mail'").SetText("jsmith@gmail.com")
      .DomRadioButton("@id='autoquote:vehicle:0'").Select()
      .DomButton("@id='autoquote:next'").Select()
      .DomTextField("@id='autoquote:age'").SetText("42")
      .DomRadioButton("@id='autoquote:gender:0'").Select()
      .DomRadioButton("@id='autoquote:type:1'").Select()
      .DomButton("@id='autoquote:next'").Select()
      .DomTextField("@id='autoquote:year'").SetText("2010")
      .DomElement("[@src='http://extjs.com/s.gif'][1]").DomClick(
        MouseButton.Left, New Point(6, 12))
      .DomElement("@textContents='Lexus'").DomClick(MouseButton.Left,
        New Point(31, 4))
      .DomTextField("@id='modelCombo'").DomClick(MouseButton.Left,
        New Point(41, 10))
      .DomElement("[@src='http://extjs.com/s.gif'][2]").DomClick(
```

```
        MouseButton.Left, New Point(14, 17))
      .DomElement("@textContents='RX400'").DomClick(MouseButton.Left,
        New Point(60, 14))
      .DomRadioButton("@id='autoquote:finInfo:1'").Select()
      .DomButton("@id='autoquote:next'").Select()
      .DomButton("@id='quote-result:purchase-quote'").Select()
      .DomLink("[@TextContents='Home'] [1]").Select()
    End With
  End With

  End Sub
End Module
```

Your script may not exactly match the preceding example. Different users interact with applications differently. For example, when filling out a form, some users click from field to field and others use the **Tab** key. SilkTest Workbench records these actions differently, though they achieve the same results. Your script should play back correctly regardless of these differences.

Playing Back the Recorded Script

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

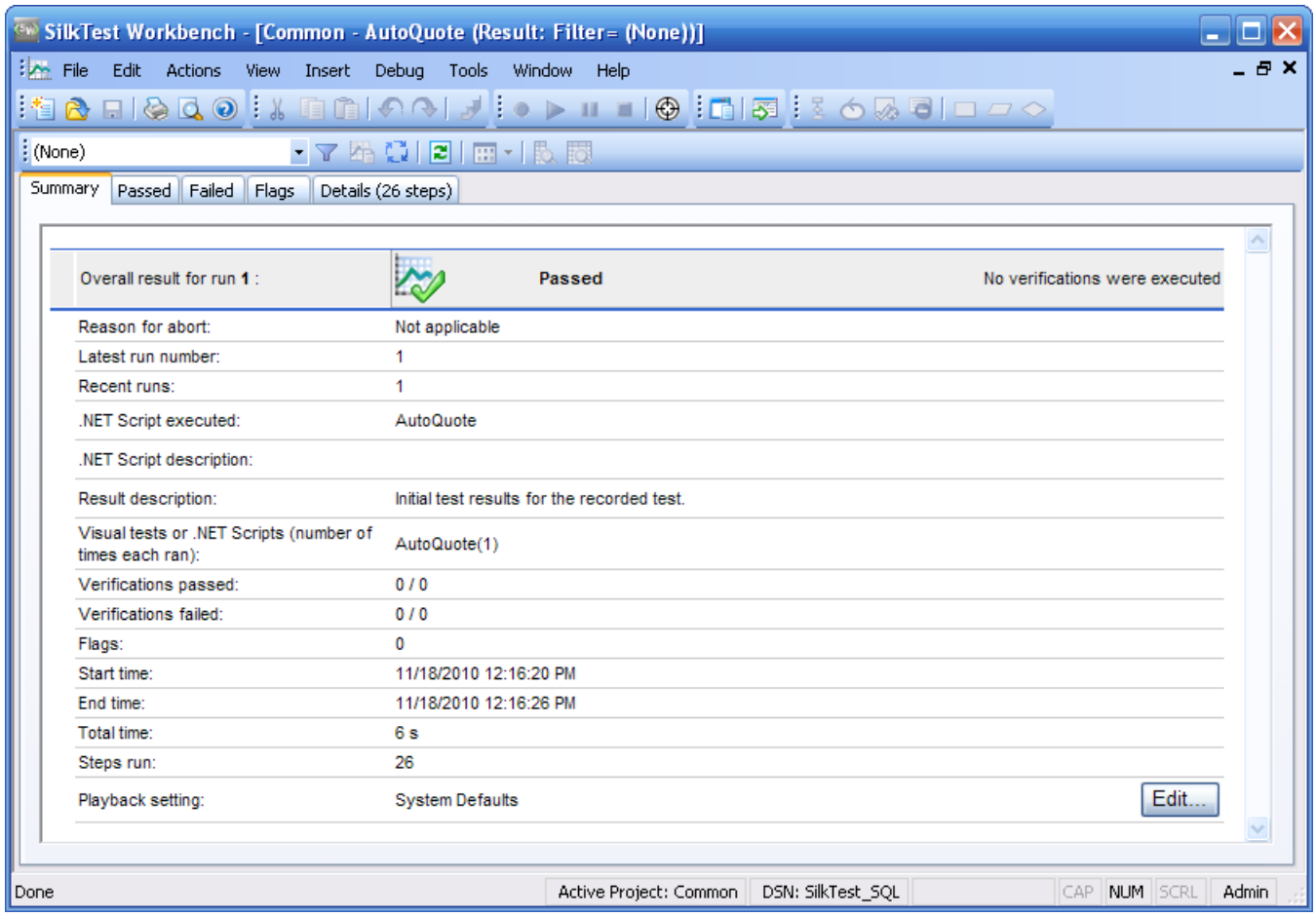
   SilkTest Workbench minimizes and the script plays back. During playback, the actions you performed while recording the script are played back on the screen against the sample application. When playback completes successfully, the **Playback Complete** dialog box opens.
4. Click **Go to Result**.
   The **Result** window opens.

Analyzing Results: Introduction

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

Result Window Overview

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 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.

Enhancing the Script: Introduction

Enhancing a test includes making updates to the existing 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 test or to remind yourself or others about important aspects of the test, you can insert a message box.

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

Inserting a Verification

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

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

1. Select the following text that defines the model type for the auto quote.
   ```
   .DomElement("@textContents='RX400'").DomClick(MouseButton.Left,
        New Point(60, 14))
   ```
2. Navigate to the page in the instant quote wizard where the Model type is specified and note a different model type.
   For example, GS430 is a model type for the Lexus make.
3. In SilkTest Workbench, change the model type.
   Change the following textContents code from:
   ```
   .DomElement("@textContents='RX400'")
   ```
   to:
   ```
   .DomElement("@textContents='GS430'")
   ```
4. To compare the expected value with the actual value and add a comment, type:
   ```
   Workbench.Verify ("GS430", "GS430", "The model type is correct")
   ```
   This condition compares the model type to the type selected. Your code should look similar to the following example:
   ```
   .DomElement("@textContents='GS430'").DomClick(MouseButton.Left,
        New Point(60, 14))
   Workbench.Verify ("GS430", "GS430", "The model type is correct")
   ```

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

Creating and Storing Application Data in a Local Variable

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

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 *stremailAddr*.

1. In the script, navigate to the email value.

   The code should look similar to the following:

   ```
   .DomTextField("@id='autoquote:e-mail'").SetText("jsmith@gmail.com")
   ```

2. Insert the following code following the email value:

   ```
   Dim StremailAddr As String
       StremailAddr = .DomTextField("@id='autoquote:e-mail'").Text
   ```

   This step creates a new local variable, *StremailAddr*, that stores the email address text from the **Get Instant Auto Quote page** page to the local variable.

3. To include output that displays the variable text during playback, include a `Console.Write` command in your script.
   For example, include:

   ```
   Console.Write (StremailAddr)
   ```

4. To view the console output, choose **View ➤ Output**.
   The **Output** window opens. When you playback a script, the **Output** window is populated.

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 Script

Now that you have made several enhancements to the recorded script, play back the script 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 script`**.
3. Click **OK**.
   SilkTest Workbench plays back the enhanced script.
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 script 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 action.
7. In the **Output** window, scroll down to the line that shows the result of storing the email address to a variable.

The text is similar to the following:

```
jsmith@gmail.com
```

8. Click the **Passed** tab to see the results of the verification for the model type.

   The text is similar to the following:

   ```
   Main:Verify Passed: The model type is correct
   ```

   The verification results also display in the **Result** and **Result Detail** columns.

Congratulations! You have successfully created a script 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 script within another script.

Executing a Script Within a Script: Introduction

In this tutorial, you created a single script that performs every action required to receive an auto insurance quote from the web application. A single script 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 scripts whenever the test application changes.

To provide an efficient means for solving these testing challenges, SilkTest 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 script in other scripts that require the same set of actions.

Modular Testing

Before creating visual tests, scripts, and other SilkTest 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 Script

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

1. Choose **File ➤ New**.
   The **New Asset** dialog box opens.
2. Select **.NET Script** from the asset types list, and then type a name for the script 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 script 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**.
   SilkTest 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 script opens in the **Code** window.

Inserting One Script Within Another

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

Executing scripts within scripts is a powerful method for efficiently testing the same basic actions in scripts.

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

1. Choose **File ➤ Open**.
   The **Asset Browser** opens.
2. Select **.NET Script** in the left pane to display the list of scripts.
3. From the list, double-click **`AutoQuote`** to open it.
   `AutoQuote` is the first test that you created in this tutorial.
4. In the **Code** window, position the cursor after the `Public Sub Main()` code, press **`Enter`** to add a new line, and type:
   ```
   Workbench.RunScript ("AddAccount")
   ```
   where *`AddAccount`* is the name of the second script that you created.
   Because we want to add the account information before we execute the quote steps, we added the `Workbench.RunScript` command before the `With` statement. To execute the *AddAccount* script after the quote steps, add the `Workbench.RunScript` command after the `End With` statement.

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 workflow. Quickly diagnosing and fixing these errors minimizes test maintenance and allows for a more efficient team testing effort.

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

Playing Back the Modular Script

In the previous lesson, you created a modular script by inserting AddAccount into the AutoQuote script.

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

1. With the AutoQuote script 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 script, the email address already exists in the database and the test fails.

Reviewing the Result

Review the results of the script.

1. Click **End** to stop playback.
   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 script, AddAccount(1).
3. Click the **Details** tab.
4. 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: The **Failed** tab does not display steps containing playback errors. It only displays failed verifications.

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

Modifying the Script that Contains Errors

The errors within our script 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 script, 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 script that resets the database and then call the new script from the AddAccount script. Creating a modular script enables you to use the Reset Database steps within other scripts as well.

1. In Internet Explorer, navigate to the **Home** page of the sample Web application.

   Starting from the **Home** page ensures that the test begins in the correct playback state.
2. In SilkTest Workbench, choose **File ➤ New**.
   The **New Asset** dialog box opens.
3. Select **.NET Script** from the asset types list, and then type a name for the script in the **Asset name** text box.
   For this tutorial, type **`ResetDatabase`** for the name.
4. Check the **Begin Recording** check box to start recording immediately.
5. Click **OK** to save the script as an asset and begin recording.
   The **Select Application** dialog box opens.
6. Select **InsuranceWeb: Home - Windows Internet Explorer** from the list and then click **OK**.
   SilkTest Workbench minimizes and a **Recording** dialog box opens.
7. From the **Home** page of the sample application, click **Settings** in the bottom right portion of the page.
8. 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.
9. Click **Home** near the top of the page to return to the home page where recording started.

10. Click **Stop Recording** and then click **Go to .NET Script**.
    The script displays in the **Code** window.
11. Click **Save**.
12. Open the AddAccount script.
13. In the **Code** window, position the cursor after the `Public Sub Main()` code, press **Enter** to add a new line, and type:

    ```
    Workbench.RunScript ("ResetDatabase")
    ```

    where *ResetDatabase* is the name of the second script that you created.
14. Open the AutoQuote script and click **Playback** on the toolbar.
    All three tests complete without any errors.

Index