Borland
A MICRO FOCUS COMPANY

# Silk Performer 10.0

# Browser-Driven Web Load Testing Tutorial

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



**2013-06-17**

# Contents

# Browser-Driven Web Load Testing Tutorial

This tutorial will assist you in the process of using Silk Performer to load-test Web 2.0 applications, especially those that rely on AJAX technologies, and get you up and running as quickly as possible. It will help you take full advantage of Silk Performer's ease of use and leading-edge functionality that is embodied in e-business' load-testing tool of choice.

## Browser-Driven Load Testing Overview

In addition to facilitating testing of today's modern Web applications on the protocol level (HTTP), Silk Performer now enables you to use real Web browsers (Internet Explorer) to generate load. In this way, you can leverage the AJAX logic built into Web applications to precisely simulate complex AJAX behavior during testing. This powerful testing approach provides results that reflect real-world end user browsing experience, including rendering time and protocol-level statistics.

Unlike other load-testing solutions that only support specific AJAX frameworks (and of those, only specific versions or a subset of controls), Silk Performer supports the full range of Web applications that are developed for (and tested with) Internet Explorer.

Please note that there are certain compatibility issues with Internet Explorer: When Internet Explorer (IE) 9.0 is installed, only the setting for IE7 Standard mode (7000) works correctly. The registry settings are correctly written, but not used by the IE-control inside Silk Performer. Internet Explorer 10 is currently not supported and when used, Silk Performer will log an error message. Note that the internal version of Internet Explorer 10 is "9.10", which is what will be displayed in the error message. There is no workaround available for this issue. However, when Internet Explorer 8 is installed, IE8 and IE7 modes can be used.

## Support for Pop-Up Windows

Silk Performer browser-driven testing supports sites that utilize pop-up windows (for example, login dialog boxes). Pop-up browser windows often include input fields in which users enter values that are passed back to the main page (for example, username and password strings). Multiple browser-window support is available by default when you create a Silk Performer project of type `Web browser-driven (AJAX)`.

A new tab is created in the Browser Application each time a pop-up window is generated during application recording. Each pop-up window that is encountered results in a tab being created in the Browser Application. Each time you click a tab in the Browser Application during recording a `BrowserActivateWindow` function is scripted automatically.

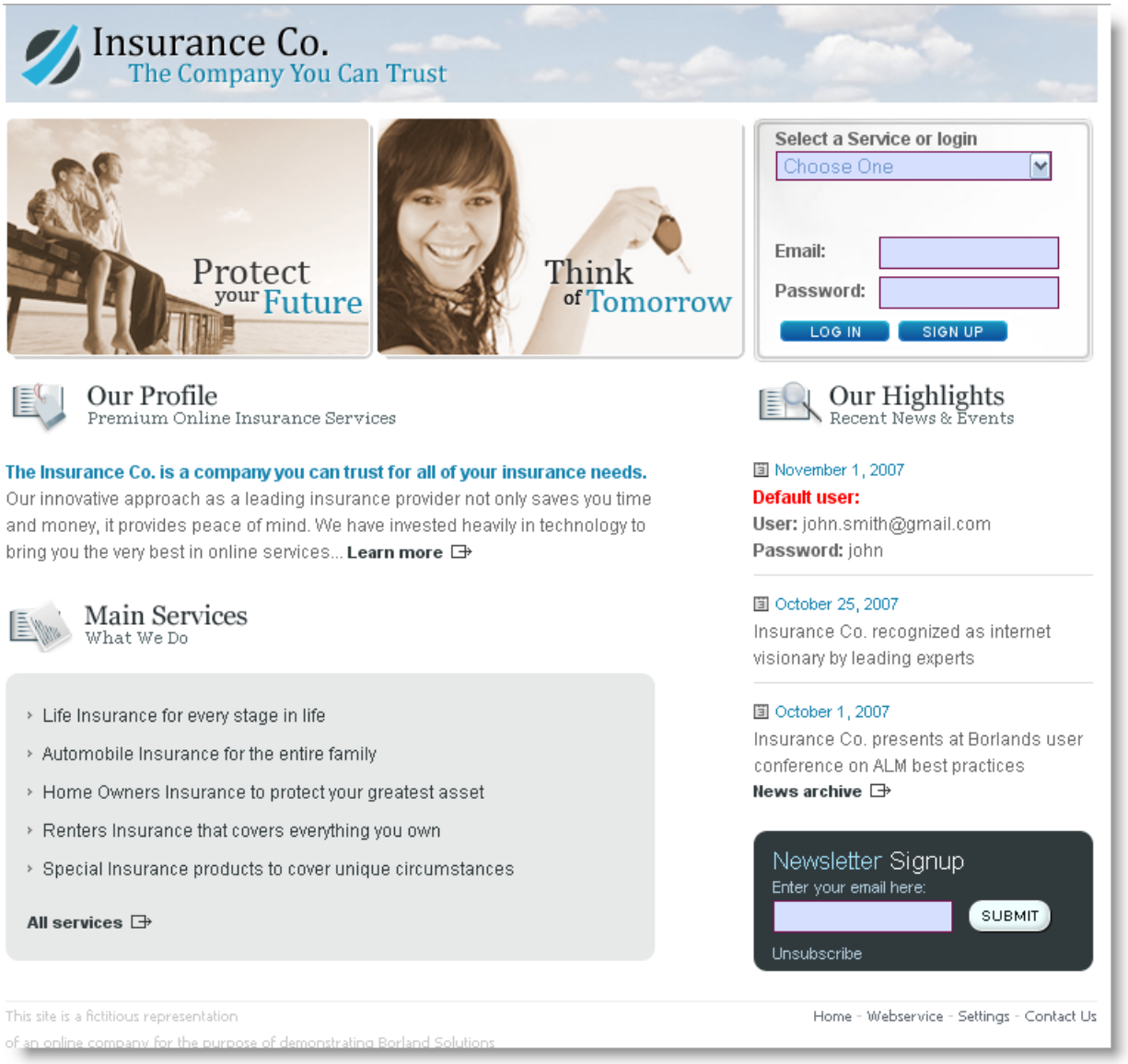
**Note:** When a single user action results in the generation of multiple browser windows, only the last generated window is recognized by the Browser Application. In other words, only the last created window results in the scripting of a `BrowserGetActiveWindow` function. None of the earlier created windows are accessible in the Browser Application.

**Note:** The manual opening of windows and tabs during recording (via menu bars, context menus, or keyboard shortcuts) is not supported.

## Sample Web 2.0 Application

Silk Performer offers a modern sample Web application that you can use to learn about Web 2.0 application testing. The InsuranceWeb sample Web application is built upon ExtJS and JSF frameworks, uses AJAX technology, and communicates via JSON and XML.

The sample application is hosted at *http://demo.borland.com/InsuranceWebExtJS/*.




# Pop-Up Window in the Sample Application

The sample Web 2.0 application includes pop-up window functionality that you can use to experiment with Silk Performer support for multiple browser windows.

1. To generate the pop-up window, visit the sample Web 2.0 application at *http://demo.borland.com/InsuranceWebExtJS/*.
2. From the **Select a Service or Log in** drop list, select **Agent Lookup**.
3. On the **Find an Insurance Co. Agent** page, click the **Open in new window** link at the bottom of the page. The **Find an Insurance Co. Agent** page loads in a new tab within the Browser Application.

Click the **Close Window** link at the bottom of the page to close the tab.

# Support for HTML Dialog Boxes

Browser-driven Web load testing does not support dialog boxes that display HTML content created by JavaScript methods `showModalDialog` or `showModelessDialog`.

When the creation of such a dialog box is detected during *recording*, the following actions are performed:

- Creation of the HTML dialog box is intercepted and suppressed
- Message is displayed to the user, indicating that HTML dialog boxes are not supported
- A comment is added to the script noting the suppression of the dialog box

When the creation of such a dialog box is detected during *replay*, the following actions are performed:

- Creation of the HTML dialog box is intercepted and suppressed
- A warning is logged, indicating that HTML dialog boxes are not supported

Silk Performer recognizes the following window types:

- Standard browser windows/tabs [fully supported]
- JavaScript dialog boxes [supported, except for the **Printer** dialog box]
- Download dialog box [fully supported]
- Modal windows, modeless windows (HTML dialog boxes) [not supported]
- Windows with embedded Active-X controls for rendering documents [not supported]

## Native Replay

Silk Performer ensures reliable script replay for browser-driven testing projects by using Windows API-level events instead of JavaScript events for the functions that perform the most frequently used UI interactions. With *native replay* (enabled by default), if the following functions are encountered in a script during replay they execute the native equivalent instead:

- A `BrowserClick` function is replayed as `BrowserNativeClick`
- A `BrowserDoubleClick` function is replayed as `BrowserNativeDoubleClick`
- A `BrowserSetText` and `BrowserSetPassword` function is replayed as `BrowserTypeKeys`
- A `BrowserMouseMove` function is replayed as `BrowserNativeMouseMove`

If replay is unable to perform a native call, for example if no mouse position can be determined to click the element, then the original scripted call is used as fallback and a warning message is logged.

Automatic native replay can be turned off by going to profile settings (**Replay - Web (Browser Driven)** > **General** > **Input** > **Legacy input mode** or by inserting `BrowserSetOption(BROWSER_OPT_LEGACY_INPUT_MODE, true)` into the test script.

Legacy input mode is turned on by default automatically for all project profiles created using Silk Performer version 9.0 or earlier.

# Web Browser Configuration Settings

Several browser settings are critical to maintaining stable test executions. Although Silk Performer works without changing any settings, there are several reasons why you may want to change these browser settings in Internet Explorer.

Please note that there are certain compatibility issues with Internet Explorer: When Internet Explorer (IE) 9.0 is installed, only the setting for IE7 Standard mode (7000) works correctly. The registry settings are correctly written, but not used by the IE-control inside Silk Performer. Internet Explorer 10 is currently not supported and when used, Silk Performer will log an error message. Note that the internal version of Internet Explorer 10 is "9.10", which is what will be displayed in the error message. There is no workaround available for this issue. However, when Internet Explorer 8 is installed, IE8 and IE7 modes can be used. Also, depending on the application under test, different browsers may script differing test scripts. If this is the case, make sure to replay a test script using the same browser as was used for recording the script.

- Increase replay speed
  - Use `about:blank` as the home page, rather than a slowly loading Web page

- Avoid unexpected browser behavior
  - Disable pop-up windows and warning dialog boxes
  - Disable auto-complete features
  - Disable password wizards
  - If Silk Performer runs on a Windows Server operating system, disable Internet Explorer Enhanced Security Configuration (IE ESC).
- Prevent browser malfunctions
  - Disable unnecessary third-party plug-ins

The following table explains where you can find these settings within the Internet Explorer GUI.

**Note:** Browser settings are located at **Tools** > **Internet Options**.

| Tab Name | Option | Configuration | Comments |
|---|---|---|---|
| General | Home page | Set to `about:blank` | Minimizes start-up time of new tabs. |
| General | Tabs | • Disable warning for closing multiple tabs<br>• Enable switch to new tab when tabs are created | • Avoids unexpected dialog boxes<br>• Links that open new tabs may not otherwise replay correctly |
| Privacy | Pop-up blocker | Disable pop-up blocker | Ensures that your Website can open new windows. |
| Content | Auto Complete | Turn off | • Avoids unexpected dialog boxes<br>• Avoids unexpected data input while typing |
| Programs | Manage add-ons | Only enable required add-ons | • Third-party add-ons may contain defects<br>• Third-party add-ons may be incompatible |
| Advanced | Settings | • Disable **Automatically check for Internet Explorer updates**<br>• Enable **Disable script debugging** (Internet Explorer)<br>• Enable **Disable script debugging** (other)<br>• Disable **Display notification about every script error**<br>• Disable all **Warn...** settings | Avoids unexpected dialog boxes.<br>**Note:** Depending on your browser version, not all settings may be available. |

# Running Multiple Virtual Users

Unlike other load testing tools, Silk Performer uses an Internet Explorer ActiveX control to simulate virtual users. The default behavior of the Internet Explorer control is to maintain a single cookie database, cache, and history for each Windows user. For load tests, Silk Performer reconfigures an Internet Explorer control

to maintain one cookie database, cache, and history for each virtual user, which is a requirement for accurate simulation.

As each virtual user has its own independent Internet Explorer control sandbox, it is possible to accurately simulate first-time and revisiting user behavior, as is used in the protocol-based approach to Web simulation.

# Defining a Browser-Driven Web Load Test Project

1. Click **Start here** on the Silk Performer workflow bar.

   **Note:** If another project is already open, choose **File** > **New Project** from the menu bar and confirm that you want to close your currently open project.

   The **Workflow - Outline Project** dialog box opens.
2. In the **Name** text box, enter a name for your project.
3. Enter an optional project description in **Description**.
4. From the **Type** menu tree, select **Web browser-driven (AJAX)**.
5. Click **Next** to create a project based on your settings.

The **Workflow - Model Script** dialog box appears.

# Creating a Test Script

The easiest approach to creating a test script is to use the Silk Performer Recorder, the Silk Performer engine for capturing and recording Web traffic and generating test scripts based on the captured traffic.

The Silk Performer Recorder captures and records the traffic that moves between the client application and the server under test. When recording is complete, the Silk Performer Recorder automatically generates a test script that is based on the recorded traffic. Scripts are written in the Silk Performer scripting language, *Benchmark Description Language (BDL).*

## Recording a Test Script

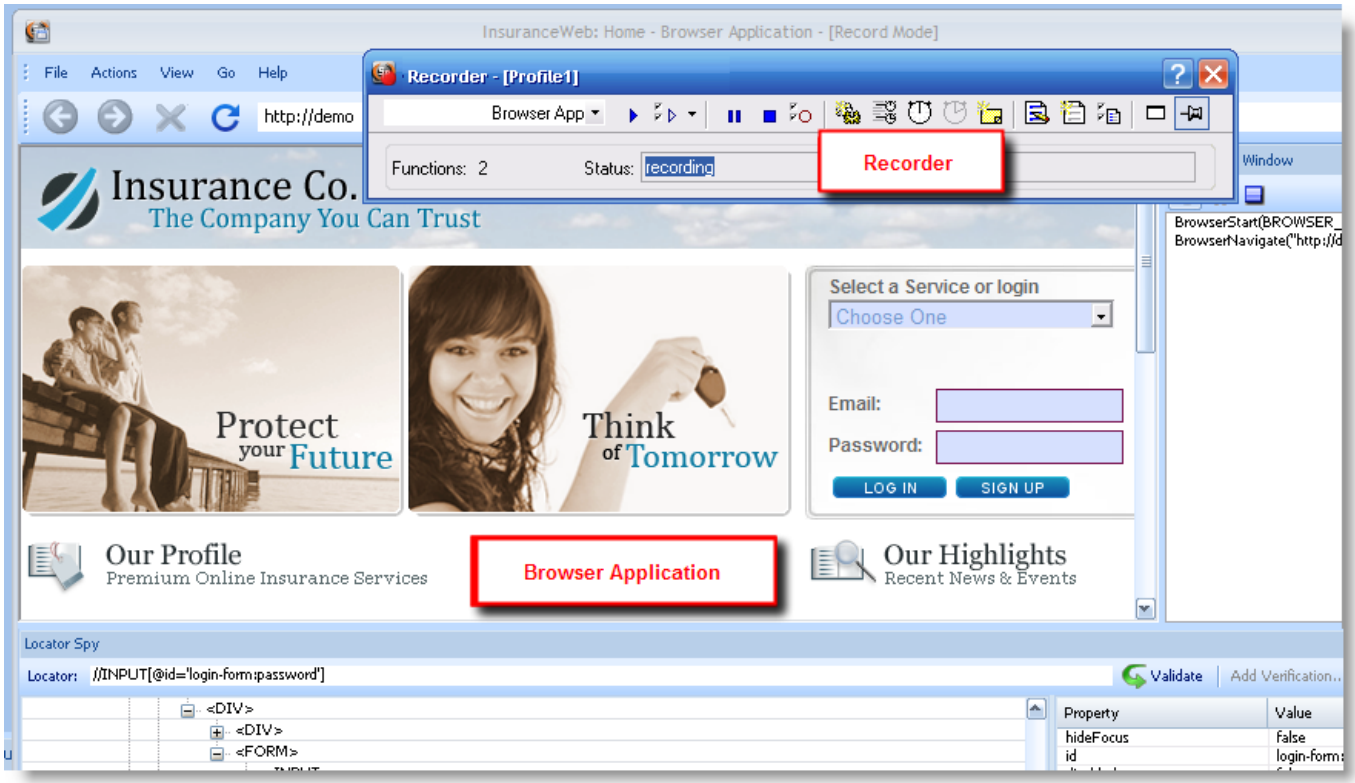
1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select Silk Performer **Browser Application** from the **Application Profile** list.
3. In the **URL** field, enter the URL that is to be recorded.

   **Note:** The InsuranceWeb sample Web 2.0 application is available at *http://demo.borland.com/InsuranceWebExtJS/*. In the **Select a Service or login** list, the `Auto Quote` and `Agent Lookup` services are available for testing while the other listed services do not provide any functionality.
4. Click **Start recording**.

   The Silk Performer recorder opens in minimized form along with the Silk Performer browser application.

   **Note:** To specify the dimensions of the browser window for recording, go to **View** > **Resize Browser Window** and define **Width** and **Height** pixel values.

   To see a report of the actions that occur during recording, maximize the recorder dialog box by clicking on the recorder toolbar.




5. Using the Browser Application Recorder, interact with the sample application in the same way that you want your virtual users to act during the test (for example, click links, type data into fields, submit data, and open the pop-up window). Your actions will be captured and recorded by the Browser Application Recorder.

   - Click (**Pause/Resume Recording**) to briefly stop and restart recording.
   - Click (**Stop Recording**) to end script recording and save your script.

6. When you are finished, close the browser window and click **Stop Recording**. The **Save As** dialog box displays.
7. Type a meaningful name for the script and click **Save.**

   A BDL test script that is based on the user actions you performed appears in the **Script** window.

# Inserting a Verification Function

1. During browser-driven script recording using the Browser Application, select a DOM object that contains a value you want to later verify during script replay (press `Pause/Break` on your keyboard to select a DOM object).

   **Note:** Tracking of UI elements must be enabled before you can select a DOM object. When tracking is enabled, a green rectangle appears around UI elements as your cursor passes over them. Click **Enable Tracking** if tracking is not currently enabled.

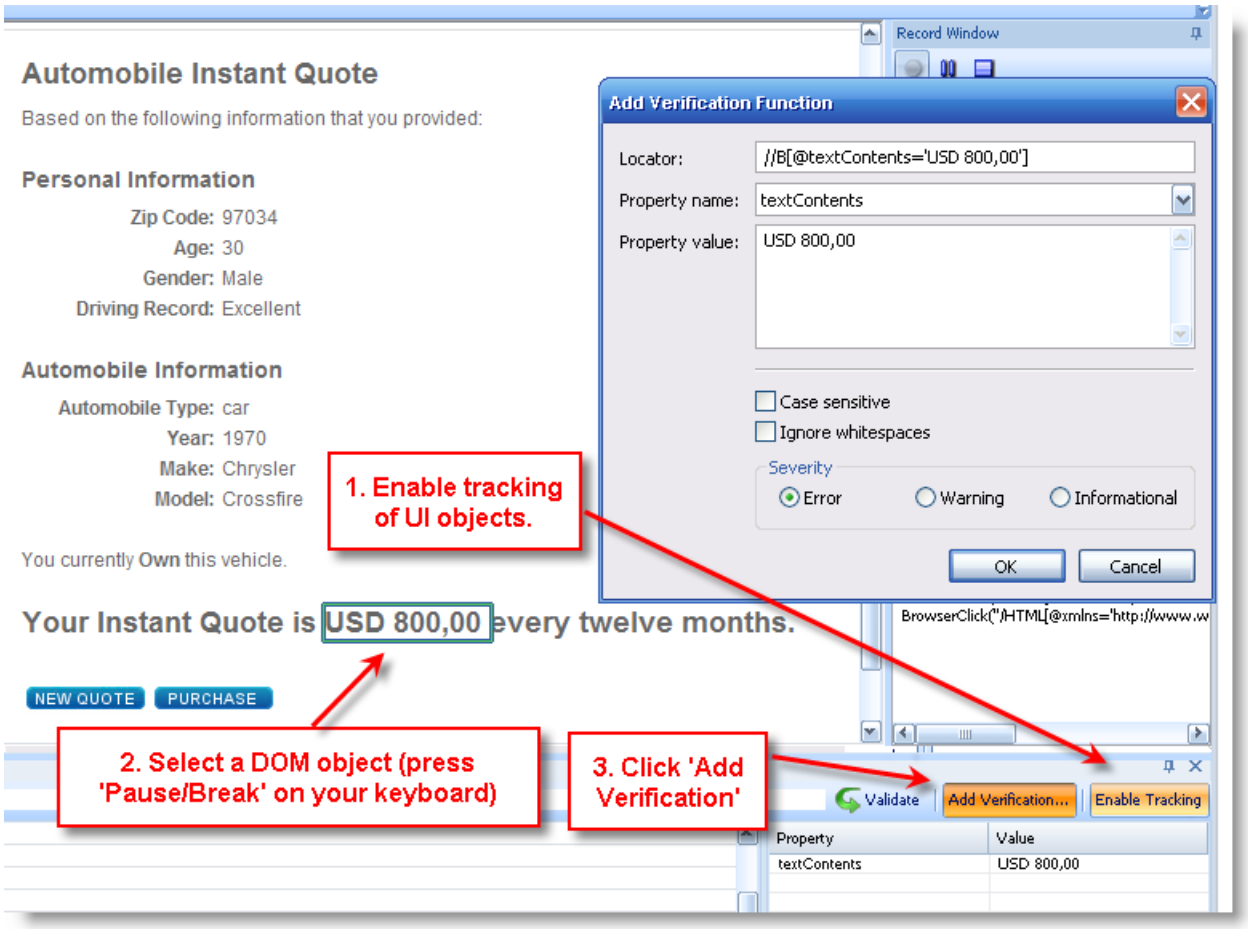
   The locator of the selected UI object appears in the **Locator** text box and the DOM hierarchy is displayed in the tree menu.
2. Click **Add Verification**.

   The **Add Verification** button is enabled when a locator value appears in the **Locator** field.

   The **Add Verification Function** dialog box appears with the locator value preloaded in the **Locator** field.
3. Select a DOM **Property name** (For example, `href`, `class`, `onmousedown`, or `textContents`).

To serve as a meaningful verification function, the selected property name should have a verifiable **Property value**. For example, property name `href` should have a property value of a specific URL.

4. Click **Okay** to insert a `BrowserVerifyProperty` verification function for the selected DOM element and its corresponding property name/value pair into the script.




The verification action is recorded in the **Record Window** and the verification function is inserted into the BDL script.

# Try Script Runs

Once you have generated a test script, determine if the script runs without error by executing a Try Script run. A Try Script run determines if a script accurately recreates the actions that you recorded with the Browser Application-Based Recorder. It also determines if the script contains any context-specific session information that you must parameterize before the script can run error free.

With Try Script runs, only a single virtual user is run and the `stress test` option is enabled so that there is no think time or delay between transactions.

**Note:** The default option settings for browser-driven Try Script runs do not include live display of content downloaded during testing (via TrueLog Explorer), though they do include the writing of log files, report files, and replay within the Browser Application **Replay window**.

# Trying Out Your Test Script

1. Click **Try Script** on the workflow bar. The **Try Script** dialog box appears with the script you created selected in the **Script** list and the active profile selected in the **Profile** list. The VUser virtual user group is selected in the **Usergroup** group box.

2. Configure settings as follows:
   a) Enable the **Visible client** option so that the Browser Application **Replay window** will display the web page content.

      Screenshots of the application state are made before each API function call.

      **Note:** Simulation settings are not applied when replaying your script with the browser application.
   b) Enable the **Step by step execution** option to run your script step by step in the **Browser Application** window.
3. Click **Run**.

   **Note:** You are not running an actual load test here, only a test run with a single virtual user to see if your script requires debugging.

   The Try Script run begins. The **Monitor** window opens, giving you detailed information about the run's progress.

## Using Step-by-Step Try Script Replay

When you enable **Step by step execution** on the **Try Script** dialog box, you are given the option of advancing your Try Script replay one step at a time.

1. Execute a Try Script run as explained above.

   Enable the **Step by step execution** option on the **Try Script** dialog box.
2. Use the buttons in the **Replay Window** to control replay:

   - Click (**Replay Step**) to execute the current API call.
   - Click (**Replay Run**) to execute the remaining API calls without further interruption.
   - Click (**Stop Replay**) to end the Try Script run.

# Common Replay Errors

Some typical reasons why scripts do not play accurately after recording are listed below. In such instances you will need to customize your test script.

- **Stateful scripts:** Recorded scripts only work when the application under test has the same state during replay that it had during script recording. For example, a script that includes user login can only be run correctly when the application is in a logged-out state. You can work around this issue by either setting the application state by manually adding logic to your script, or you can ensure that your recorded scripts do not change application state in the first place (for example, you could include user log out during the recording of your script).
- **Temporarily generated DOM attributes:** Some AJAX frameworks generate attributes that change each time a page is loaded (for example, `x-auto` values in `ext`). If a locator relies on such attributes, script replay will fail. You will need to add the attributes to the ignored attributes list to prevent them from being recorded in the future.
- **Missing mouse movements:** When you are testing websites where items only appear if you are hovering with your mouse over certain elements (for example a button or a menu item), you will get an error during the replay of the script. Silk Performer cannot detect the item because the hovering event is not recorded. Menus that are built with JavaScript are a good example for such a case. However, with Silk Performer you can fix this problem during the replay of a script. In the **Browser Application**, you can click the **Troubleshoot** button when the error occurs, select **Insert Mouse Move** from the list, move the mouse over the UI element, press **<Pause/Break>** on your keyboard, click **Insert**, and click **Rerun Script**. Now the script will run without an error.
- **Calls that run into the synchronization timeout:** Built-in AJAX synchronization waits until the browser is in an idle state before API calls are returned. This is a key factor in reliable testing of AJAX-based applications. However, in some situations there is no idle state (for example, if a page uses polling or keeps connections open for server-push events). In such situations the synchronization waits

until it runs into a timeout. You can work around this issue by temporarily setting the synchronization mode back to HTML.

# Analyzing Test Scripts

In contrast to the Web-protocol approach to load testing, browser-driven Web load testing uses the Browser Application for script validation.

The benefits of having Try Script runs performed in the Browser Application are as follows:

- Live application state is presented in the browser, including Locator Spy functionality for advanced script modification and adaption.
- Scripts can be executed in step-by-step mode.
- Screenshots are captured before each browser API call and stored in the TrueLog for future analysis.

Once a Try Script run is shown to be successful in the Browser Application, you can analyze the results of the Try Script run with TrueLog Explorer. Test script analysis with TrueLog Explorer involves the following tasks:

- Viewing Virtual User Summary Reports
- Finding errors
- Comparing replay test runs with recorded test runs

# Visual Analysis with TrueLog Explorer

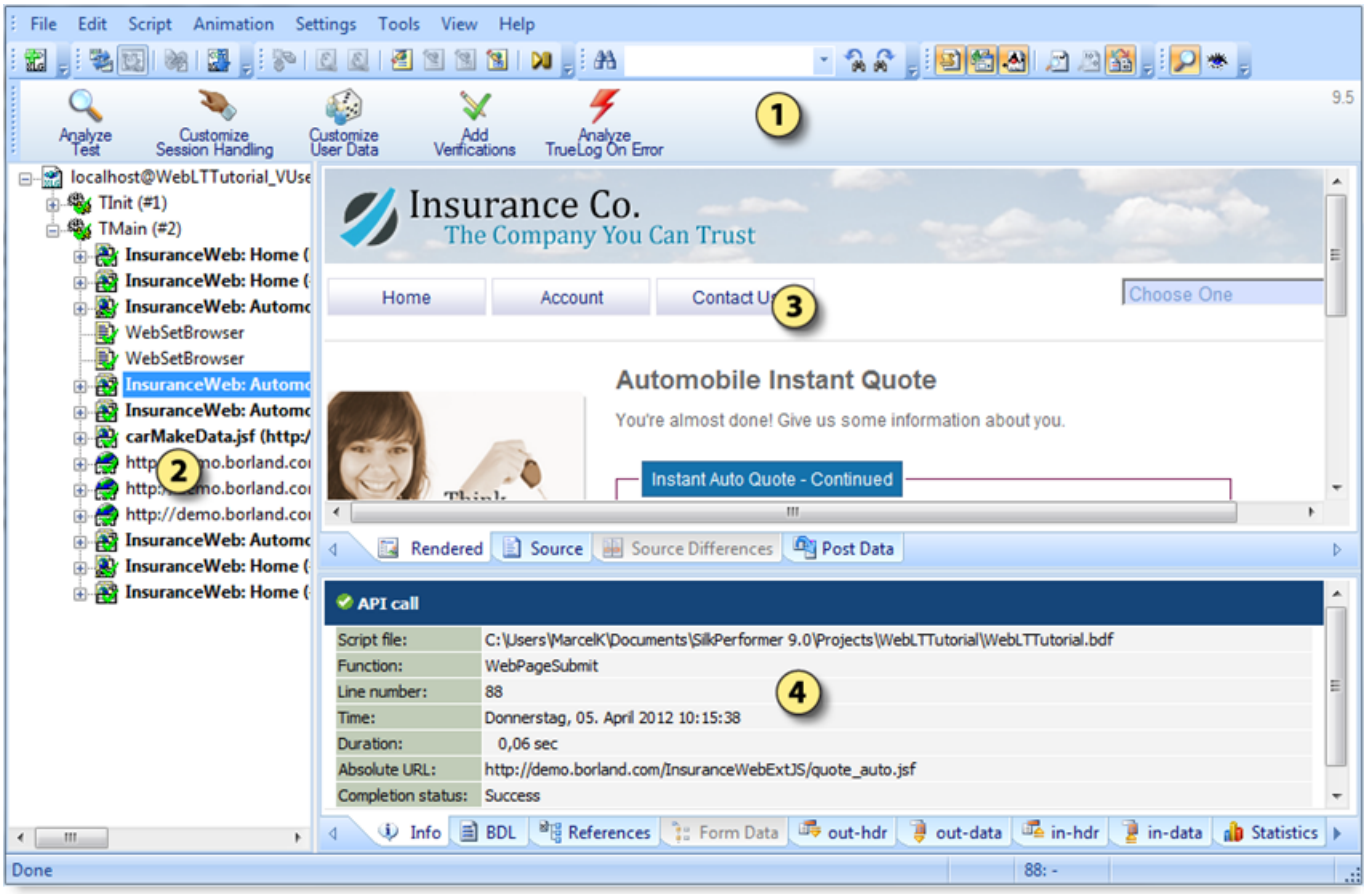
One of TrueLog Explorer's most powerful features is its ability to visually render Web content that is displayed by applications under test. In effect, it shows you what virtual users see when they interact with an application.

The TrueLog Explorer interface is comprised of the following sections:

- The **Workflow Bar** acts as your primary interface as you work with TrueLog Explorer. The Workflow Bar reflects TrueLog Explorer's built-in testing methodology by supporting its five primary tasks.
- The **API Node Tree** menu on the left of the interface allows you to expand and collapse TrueLog data downloaded during tests. Each loaded TrueLog file is displayed here along with links to all relevant API nodes. You can click a node to display a screen shot in the **Screen** pane and history details in **Information** view.
- The **Content** pane provides multiple views of all received data.
- The **Information** pane displays data regarding testing scripts and test runs, including general information about the loaded TrueLog file, the selected API node, BDL script, and statistics.

**Note:** HTTP header data is not currently available.

**Note:** To launch TrueLog Explorer from Silk Performer, choose **Results** > **Explore TrueLog**.




1 **Workflow bar**

2 **API node tree menu**

3 **Content pane**

4 **Information pane**

# Analyzing a Test Run

1. With the TrueLog from a Try Script run loaded into TrueLog Explorer, click the **Analyze Test** button on the Workflow bar.

   The **Analyze Test** dialog box displays.
2. Proceed with one of the following options:

   - View a virtual user summary report
   - Look for errors in the TrueLog
   - Compare the replay test run to the recorded test run

# Viewing a Summary Report

Virtual user summary reports are summary reports of individual Try Script runs that offer basic descriptions and timing averages. Each report tracks a separate virtual user and presents data in tabular format.

Virtual user summary reports include details regarding the following:

- Virtual users

- Uncovered errors
- Response time information tracked for each transaction defined in a test script
- Page timer measurements for each downloaded Web page
- Individual timers and counters used in scripts (Measure functions)

# Displaying a Virtual User Summary Report

1. With the TrueLog generated by your Try Script run loaded into TrueLog Explorer, click the **Analyze Test** button.
2. Click the **Show the virtual user summary report** link.

# Enabling Summary Reports

Because virtual user summary reports require significant processing resources, they are not generated by default. To enable the automatic display of virtual user reports at the end of animated TryScript runs (or by clicking the root node of a TrueLog file in the **API Node Tree** menu) enable the **Display virtual user report** option ( **Settings** > **Workspace** > **Reports** ).

**Note:** Virtual user reports can also be viewed within Silk Performer by right-clicking a virtual user name and selecting **Show Virtual User Report File**.

# Finding Errors in a TrueLog

TrueLog Explorer helps you find errors quickly after Try Script runs. Erroneous requests can be examined and necessary customizations can be made via TrueLog Explorer.

**Note:** When viewed in the **API Node Tree** menu, API nodes that contain replay errors are tagged with red “X” marks.

1. With the TrueLog generated by your Try Script run loaded into TrueLog Explorer, click the **Analyze Test** button.
2. Click the **Find errors** link. The **Step through TrueLog** dialog appears with the **Errors** option selected.
3. Click **Find Next** to step through TrueLog result files one error at a time.

# Viewing Page Statistics

After verifying the accuracy of a test run, you can analyze the performance of your application under “no-load” conditions via page statistics.

Overview pages detail:

- Action time: Total page response times, including processing and rendering in the browser.
- Documents time: Document download times (including server busy times), and time elapsed for receipt of embedded objects.

Detailed action statistics show exact response times for individual Web page components, allowing you to easily pinpoint the root causes of errors and slow page downloads.

Because Try Script runs do not include think times, the measurements they produce cannot be used to predict real-world performance.

Detailed action statistics include the following data for each page component:

- DNS lookup time
- Connection time
- Round-trip time
- Cache statistics

**Note:** Compared to the protocol-based approach, browser-driven test statistics do not include certain low-level/protocol-related metrics.

## Viewing an Overview Page

1. From the API Node Tree menu, select the API node for which you would like to view statistics.
2. Select **Browser Nodes** on the **Step through TrueLog** dialog box.
3. Click the **Statistics** tab to open **Statistics** view.
4. Select specific components listed in the URL column for detailed analysis and page drill-down.

## Comparing Record and Replay Truelogs

With Web application testing, TrueLog Explorer shows the actual Web pages that are received during tests. Live monitoring of downloaded data is available via TrueLog Explorer animated mode. Data is displayed as it is received during testing.

By comparing a TrueLog that has been generated during the script development process alongside the corresponding TrueLog was recorded originally, you can verify that the test script runs accurately.

1. Click the **Analyze Test** button on the Workflow Bar. The **Workflow - Analyze Test** dialog box appears.
2. Click **Compare your test run**.
3. The corresponding recorded TrueLog opens in Compare view and the **Step through TrueLog** dialog box appears with the **Browser Nodes** option selected, allowing you to run a node-by-node comparison of the TrueLogs.
4. Click the **Find Next** button to step through TrueLog result files one page at a time.

   **Note:** Windows displaying content presented during replay have green triangles in their upper left corners. Windows displaying content originally displayed during application recording have red triangles in their upper left corners.

# Configuring Project Profile Settings

Silk Performer offers a variety of browser-driven Web load-testing profile settings. Web (browser-driven) profile settings are project-specific settings that relate to synchronization and object locator generation. These settings are specified on a per-project basis.

**Note:** For the purposes of this tutorial, you do not need to change the default settings.

## Configuring Browser-Driven Recording Settings

1. Right-click the **Profiles** node in the **Project** tree menu and select **Edit Active Profile**. The **Profile - [Profile1] - Simulation** dialog box displays at the **Simulation** tab (**Replay** category).
2. Click **Record**.
3. Scroll down and select **Web (Browser Driven)**.
4. Select the **Recording** tab.
5. Type any DOM attribute names that should be ignored during recording in the **Ignored DOM attribute names** text field. Attribute names that match any pattern in the **Ignored DOM attribute names** field will be ignored during recording.
6. Type any DOM attribute values that should be ignored during recording in the **Ignored DOM attribute values** text field. Attribute values that match any pattern in the **Ignored DOM attribute values** field will be ignored during recording.

7. The **Preferred DOM attribute names** option configures the name of the custom attributes that are recorded.
8. Click **OK**.

# Configuring Browser-Driven Replay Settings

1. In the **Projects** tree menu, right-click the **Profiles** node and select **Edit Active Profile**. The **Profile - [Profile1] - Simulation** dialog box opens at the **Simulation** tab.
2. Click the **Replay** category button.
3. Scroll down to and select **Web (Browser Driven)**. The **Web (Browser Driven) / General** tab displays.
4. Use the **Simulation** group box to set options for realistic simulation of users visiting Web sites:
   - Click the **First time user** option button to generate a realistic simulation of users who visit a Web site for the first time.

     Persistent connections will be closed, the Web browser emulation will be reset, and the document cache, the document history, the cookie database, the authentication databases, and the SSL context cache will be cleared after each transaction. In such instances, Silk Performer downloads the complete sites from the server, including all files.
   - Click the **Revisiting user** option button to generate a realistic simulation of users who revisit a Web site. Non-persistent sessions will be closed, but the document history, the persistent cookie database, and the context cache will not be cleared after each transaction. In such cases, pages are not downloaded if they exist in the document cache.
   - Select the **IE Compatibility Mode** to define the rendering mode that Internet Explorer (IE) uses to display automatic replaying on the user's Web browser. You can configure the version of IE that causes sending different HTTP headers to the server and rendering of the Web content. For example, IE9 as IE7 sends IE7 headers and renders as IE7. The **Default** value depends on the user's Internet Explorer browser version.

   **Note:** Simulation settings are not applied when replaying your script with the Browser Application. However, all caching settings that you configure within Internet Explorer's Internet options will be applied to your browser-driven tests.
5. Ensure that the **Legacy input mode** setting is disabled.

   Silk Performer ensures reliable script replay for browser-driven testing projects by using Windows API-level events instead of JavaScript events for the functions that perform the most frequently used UI interactions. With *native replay* (enabled by default), if the following functions are encountered in a script during replay they execute the native equivalent instead:

   - A `BrowserClick` function is replayed as `BrowserNativeClick`
   - A `BrowserDoubleClick` function is replayed as `BrowserNativeDoubleClick`
   - A `BrowserSetText` and `BrowserSetPassword` function is replayed as `BrowserTypeKeys`
   - A `BrowserMouseMove` function is replayed as `BrowserNativeMouseMove`

   If replay is unable to perform a native call, for example if no mouse position can be determined to click the element, then the original scripted call is used as fallback and a warning message is logged.

   Legacy input mode is turned on by default automatically for all project profiles created using Silk Performer version 9.0 or earlier.
6. Click the **Synchronization** tab.
7. Configure **Synchronization** settings as required.
   - The **Synchronization mode** option configures the algorithm that is used to wait for the ready state of a browser invoke call (pre and post invocation).
   - The **Synchronization timeout** option configures the maximum time in milliseconds that is used to wait for an object to be ready (pre and post invocation).

- In the **URLs to exclude from synchronization** text box, type the entire URL or a fragment of the URL for any service or Web page that you want to exclude. Some AJAX frameworks or browser applications use special HTTP requests, which are permanently open in order to retrieve asynchronous data from the server. These requests may let the synchronization hang until the specified synchronization timeout expires. To prevent this situation, either use the HTML synchronization mode or specify the URL of the problematic request here. Separate multiple entries with a comma.
- The **Object resolve timeout** option configures the maximum time in milliseconds to wait for an object to be resolved during replay.
- The **Object resolve retry interval** option configures the time in milliseconds after which another replay attempt should be made following an object not resolving.

**8.** Click **OK**.

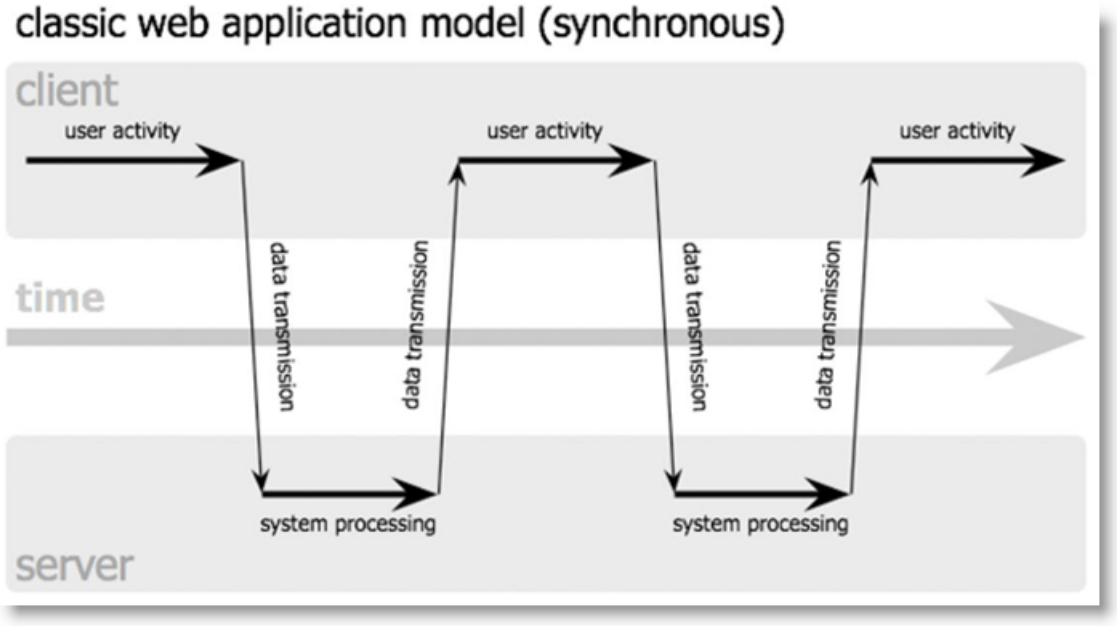
# Technology Overview

The topics in this section offer an overview of the AJAX Web application model and processing flow. They also examine the implications of AJAX on automated load testing and the use of XPATH in identifying DOM elements.

## AJAX Overview

*AJAX* (Asynchronous JavaScript and XML) is a group of interrelated Web development techniques that are used on the client-side (browser) to create interactive Web applications. With AJAX, Web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. For data encoding, typically XML or JSON formats are used, although proprietary data encoding formats are also used.

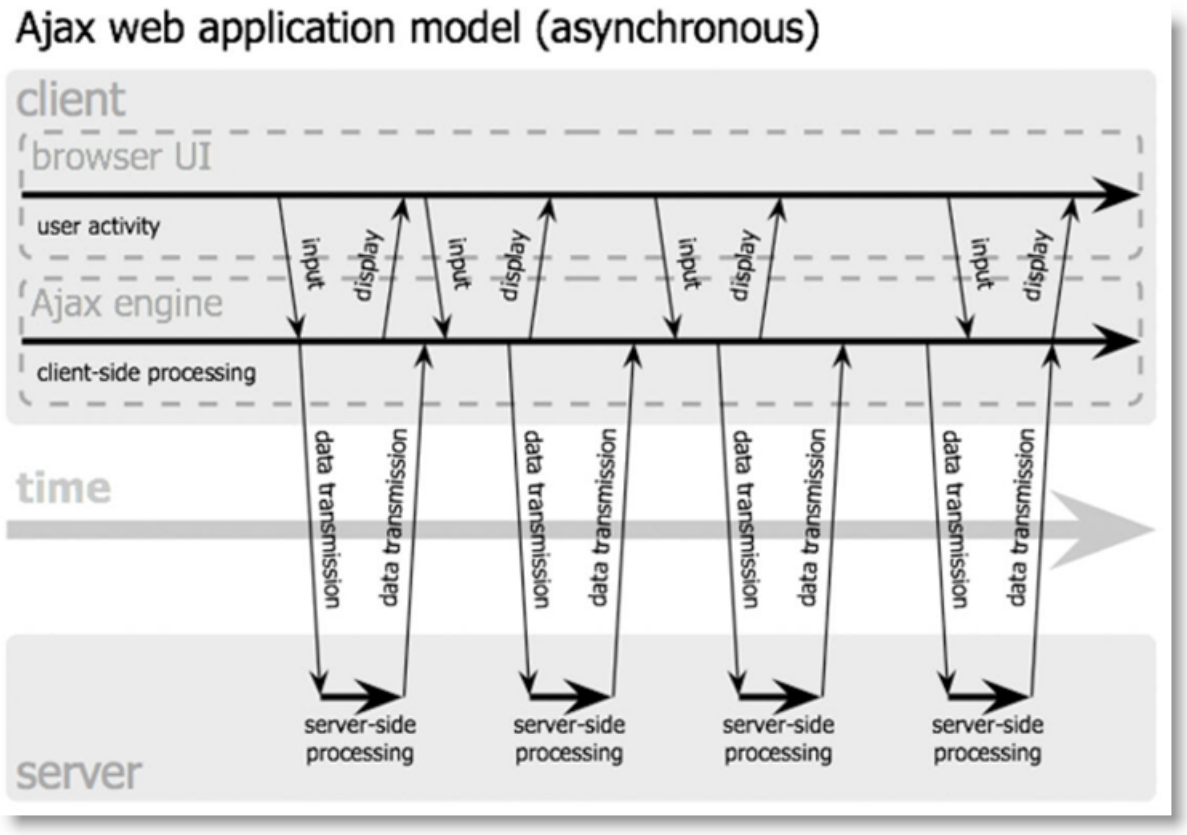
In many cases, related pages on a Web site share a lot of common content. Using traditional methods, that content has to be reloaded upon each page request.




Using AJAX, a Web application can request only the select content that is needed to update the page, thereby dramatically reducing bandwidth usage and load time.

The use of asynchronous requests allows the client Web browser UI to be more interactive and to respond quickly to inputs. In many instances, sections of pages can also be reloaded individually. Users often

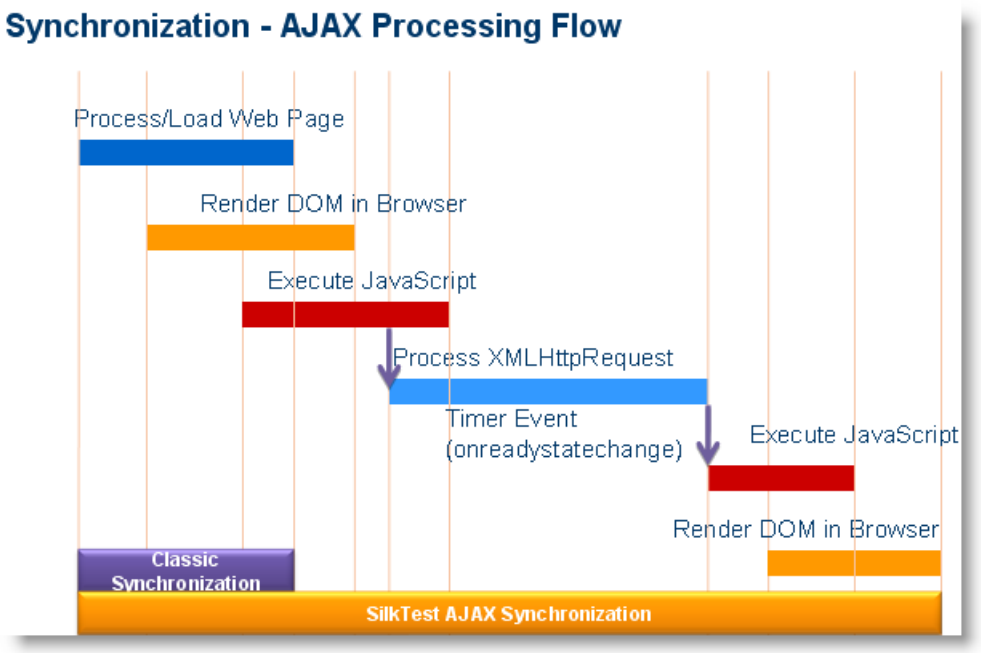
perceive such applications as being faster and more responsive, even when application state on the server-side has not changed.




Web applications are typically based on AJAX frameworks, such as Ext JS and Ext GWT, though this is not a requirement.

# AJAX Processing Flow

With Web 2.0 applications, the entire concept of *pages* becomes blurred. It is difficult to determine the time when a Web page has finished loading. The initial HTTP request is usually answered with some HTML response including additional resources that the browser loads in subsequent requests. When executing the JavaScript, additional requests may be transmitted via `XMLHttpRequest` calls. The responses to such asynchronous calls may be reflected by a change/adaption of the initial page.




Some pages are updated frequently without any user interaction. Other events that may trigger asynchronous calls are:

- Mouse-overs
- Key strokes
- Drag and drop operations
- Timers
- Network events (on `readystatechange`)

# Implications for Automation and Load Testing

As the concept of *pages* is not applicable with most AJAX applications, it can be difficult to determine when to allow sequential Web page actions. The major problem that automation and testing tools run into is synchronization: Parts of a page may be visible but not yet functional. For example, you might click a button or link and nothing happens, or incorrect behavior occurs. This is usually not an issue for a human user because they can simply perform the action again. Being slower than automation tools, users are often not even aware of such problems.

Silk Performer uses a sophisticated technique for determining when a Web page is ready for a subsequent action. This AJAX mode synchronization waits for the browser to be in an idle state. This is especially useful for AJAX applications or pages that contain AJAX components. Using the AJAX mode eliminates the need for manual scripting of synchronization functions (for example, such as waiting for an object to appear/disappear, waiting for a specific property value), which dramatically eases the script development process. This automatic synchronization is also the basis for successful record and replay that does not require manual script customization.

### Troubleshooting

Because of the true asynchronous nature of AJAX, generally there is no true browser idle state. Therefore, in some situations, it is possible that Silk Performer will not recognize the end of an invoked method call and will throw a timeout error after the specified timeout period.

# Identifying DOM Elements Using XPath

Silk Performer uses a subset of the XPath query language to identify DOM elements. For additional information about XPath, see *http://www.w3.org/TR/xpath20/*. All API calls that interact with DOM elements

take XPath query strings as input parameters. The XPath query strings are referred to as locators. For example, the API call `BrowserLinkSelect("//a[@href='www.companyxyz.com']")` uses the locator `//a[@href='www.companyxyz.com']` which identifies the first link that has `www.companyxyz.com` as the value of its href attribute (for example, `<a href="www.companyxyz.com">My Link</a>`).

The following table lists all XPath constructs that are supported by Silk Performer

| Supported XPath Construct | Sample | Description |
|---|---|---|
| `Attribute` | `a[@href='myLink']` | Identifies all DOM Links with the given href attribute that are children of the current context. All DOM attributes are supported. |
| `Index` | `a[1]` | Identifies the first DOM link that is a child of the current context. Indices are 1-based in XPath. |
| Logical Operators: `and`, `or`, `=` | `a[(@href='microfocus' or @caption != 'b') and @id='p']` | |
| `.` | `.//div[@id='menuItem']/.` | The" ." refers to the current context (similar to the well known notation in file systems. The example is equivalent to `//div[@id='menuItem']/` |
| `..` | `//input[@type='button']/../div` | Refers to the parent of an object. For example, the sample identifies all divs that contain a button as a direct child. |
| `/` | `/form` | Finds all forms that are direct children of the current object. `./form` is equivalent to `/ form` and `form`. |
| `/` | /form/input | Identifies all input elements that are a child of a form element. |
| `//` | `//input[type='checkbox']` | Identifies all check boxes in a hierarchy relative to the current object. |
| `//` | `//div[id='someDiv'//input[type='button']` | Identifies all buttons that are a direct or indirect child of a div that is a direct or indirect child of the context. |
| `/` | `//div` | Identifies all divisions that are direct or indirect children of the current context. |

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

| Unsupported XPath Construct | Example |
|---|---|
| Comparing two attributes to one another | `a[@caption = @href]` |
| Attribute names on the right side are not supported. Attribute names must be on the left side. | `a['abc' = @caption]` |
| Combining multiple XPath expressions with `and` or `or`. | `a [@caption = 'abc']` or `.//input` |
| More than one set of attribute brackets | `div[@class = 'abc'] [@id = '123']` (use `div[@caption = 'abc' and @id = '123']` instead) |

| Unsupported XPath Construct | Example |
|---|---|
| More than one set of index brackets | `a[1][2]` |
| Wildcards in tag names or attribute names | `*/@c?ption='abc'` |
| Logical operators: `not`, `!=` | `a[@href!='someValue']`, not`[@href='someValue']` |
| Any construct that does not explicitly specify a class or the class wildcard. For example, including a wildcard as part of a class name. | `//[@caption = 'abc']` |

# Browser Application and Locator Spy Usage

To enable convenient record/replay, Silk Performer provides its own Browser Application. The application offers the following features:

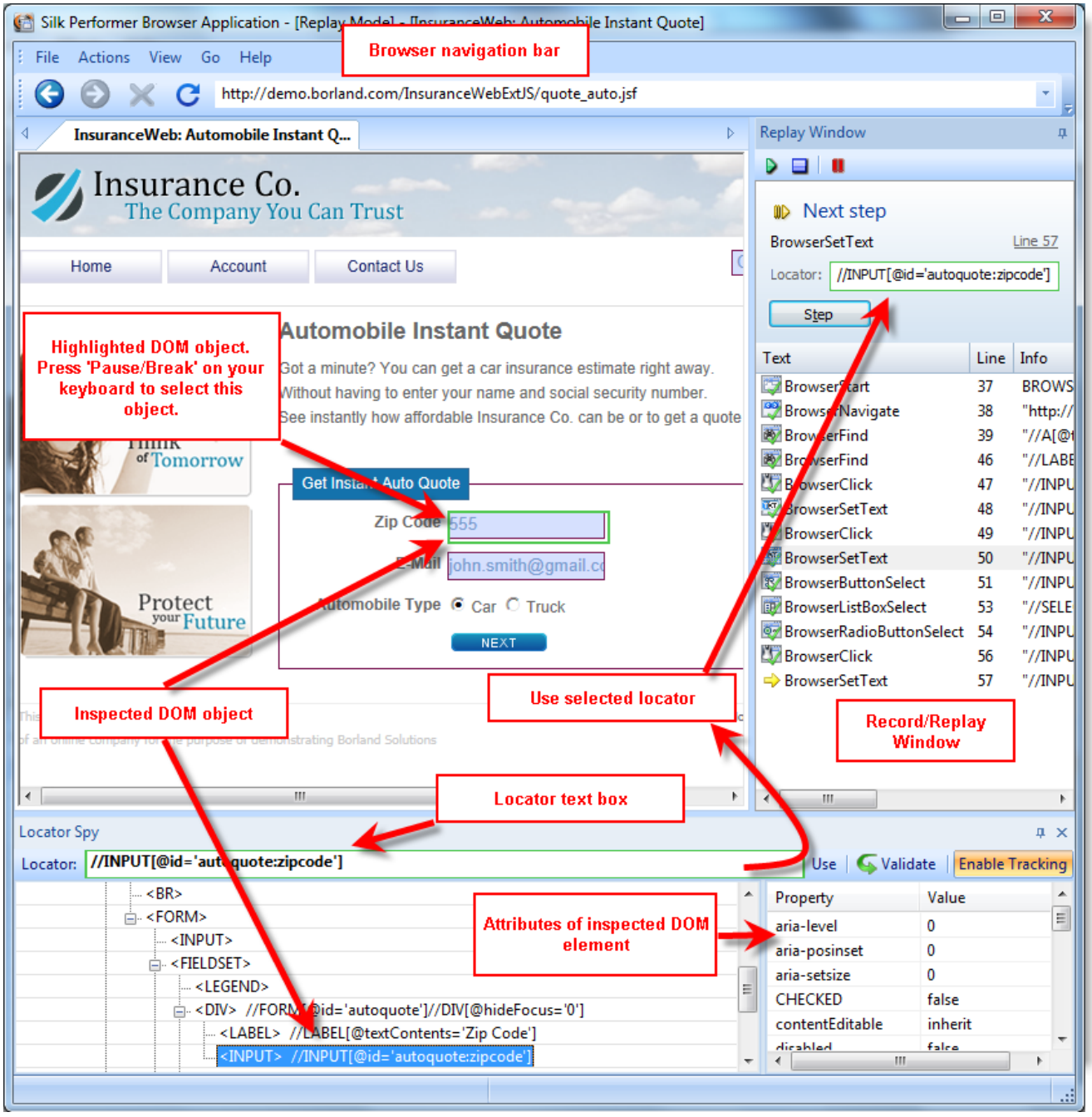
- **Browser Window**
- **Locator Spy**
- **Record/Replay Window**

The **Record/Replay Window** displays logging information during both record and replay. It allows you to start/stop and pause/resume recording (during record mode) and to pause/resume replay (during replay mode).

The following image shows the Silk Performer Browser Application in replay mode and identifies the most important elements of the browser window and the Locator Spy.

**Note:** Tracking of UI elements must be enabled before you can select a DOM object. When tracking is enabled, a green rectangle appears around UI elements as your cursor passes over them. Click **Enable Tracking** if tracking is not currently enabled.

The locator of the selected UI object appears in the **Locator** text box and the DOM hierarchy is displayed in the tree menu.




### Browser Navigation Bar

The **Browser navigation bar** enables standard browser navigation.

### Highlighted DOM Element

On mouse moves, the DOM element under the current mouse position is determined and the position of the DOM element is indicated by a green rectangle. This enables you to get a feeling for the architecture of the current page and the hierarchy of its DOM elements.

### Inspected DOM Element

Pressing `Pause/Break` triggers the following actions:

- The highlighted DOM element becomes the inspected DOM element.
- The position of the inspected DOM element is indicated by blue highlighting.
- The DOM hierarchy tree of the current page is determined and displayed in the **Locator Spy** by the HTML tags of the DOM elements.
- The path to the inspected DOM element is expanded and the inspected DOM element is selected.
- The attributes of the inspected DOM element are determined and displayed.
- The locator for the inspected DOM element is determined and displayed in the **Locator edit field**.

To search through a selected DOM object for a specific text or numerical string, press `Ctrl+F` on your keyboard to open the **Find in Locator Spy** dialog box (alternatively, select **Actions** > **Find in DOM Tree**). You can search for strings within **Tags**, **Property names**, or **Property values**. Click **Next** to step through all instances of the search string.

To change the inspected DOM element, press `Pause/Break` on any highlighted DOM element or select another DOM element within the DOM hierarchy tree.

When selecting another DOM element from within the DOM hierarchy tree, the locator for the DOM element is determined and displayed next to the DOM element's HTML tag. In addition to updating the tree item text, the **Locator** text box is updated and the position of the DOM element on the current page is indicated by blue highlighting.

When a page's DOM becomes invalid after pressing `Pause/Break` and the locator for the newly selected DOM element can not be found, a red border is displayed around the **Locator** text box. By pressing `Pause/Break`, the hierarchy tree is refreshed and the current DOM object is highlighted. Locator strings in the DOM hierarchy tree are also removed as they are now invalid.

#### Locator Text Field

The **Locator** text field shows the locator string of the currently inspected DOM element. Whenever the inspected DOM element changes, the locator is updated.

The text field may be used to copy a locator string to another location, for example to a BDL script, or to manually edit locator strings for validation of user defined locators on the current page. While editing the locator string, click **Validate** to validate the locator. In the case of successful validation, the position of the DOM element corresponding to the locator string is highlighted in green. In the case of unsuccessful validation, the border of the**Locator** text field is highlighted in red.

If you want to add a verification during a try script run, pause the replay and click **Add Verification**. Adding verifications during a try script run works exactly as during recording.

In the right window of the **Locator Spy**, you can right-click a property and copy the property name, the property value, or both to the clipboard. If you copy both, the string will be saved in the form `@name='value'`. A real-world example is `@hideFocus='false'`. This offers a convenient method of exchanging a property in the **Locator** field.

#### Attributes of Inspected DOM Element

This is a list of attributes (name/value pairs) belonging to the current inspected DOM element. If the default generated locator string does not meet the requirements, build a manually edited locator string using some of the listed attributes.

# Locator Verification in Browser Application

The Browser Application offers commands that make it easier to analyze and navigate locator information in the **Replay** window. Right click any API call in the **Replay** window to access context-sensitive commands for copying that call's locator information, copying the content of the **Info** column, and displaying the locator of the call in the **Locator Spy** DOM hierarchy tree.

Such commands can be useful when, for example, a locator verification or an API call fail. You can use the locator of the API call to locate the call in **Locator Spy**, troubleshoot the issue, and edit the script

accordingly. You can also use the **Copy** command to copy and paste API details into emails and issue reports.

## Inserting Mouse Move

When you are testing websites where items only appear if you are hovering with your mouse over certain elements (for example a button or a menu item), you will get an error during the replay of the script. Silk Performer cannot detect the item because the hovering event is not recorded. Menus that are built with JavaScript are a good example for such a case. However, with Silk Performer you can fix this problem during the replay of a script.

In the **Browser Application**, you can click the **Troubleshoot** button when the error occurs, select **Insert Mouse Move** from the list, move the mouse over the UI element, press **<Pause/Break>** on your keyboard, click **Insert**, and click **Rerun Script**. Now the script will run without an error.

# Defining Browser Window Dimensions for Recording

Launch the Browser Application for browser-driven load testing .

**Note:** Browser dimensions can only be defined during script recording.

1. To define specific browser-window dimensions for recording, go to **View** > **Resize Browser Window**. The **Resize Browser Window** dialog box is displayed.
2. Specify a **Width** setting (in pixels).
3. Specify a **Height** setting (in pixels).
4. Click **OK**.

# Troubleshooting Browser-Driven Load Testing Issues

Learn how to start the perfrun process using an actual user account, handle client certificates, and exclude specific URLs from AJAX synchronization.

**Note:** Browser-driven load testing is supported for Internet Explorer 7, 8, 9, 10.

## Browser-Driven Virtual Users on Remote Agents

Starting a remote agent with an actual user account rather than the system account, which is the default, makes a big difference for browser-driven virtual users. Each virtual user employs its own Internet Explorer instance, which loads the settings stored in the Microsoft Windows user's profile.

Under the system account, Internet Explorer loads different settings than under a user account. Typically Internet Explorer utilizes fewer or different HTTP headers than with user accounts. In order to avoid the issue of recorded traffic differing from generated traffic, it is recommended to run remote agents under a user account.

**Note:** Ensure that the specified user account is a member of the Remote Desktop Users Windows group on the remote agent.

The required account setting can be configured in System Configuration Manager on the **Applications** tab or a user account can be set in the **System Settings** > **Agents** > **Advanced** tab if all remote agents should run under the same user account.

# Handling Client Certificates

You can select a client certificate during script recording. Client certificates facilitate authentication against certain Web sites. APIs are now available for importing certificates to and deleting certificates from the Microsoft certificate store, which is used by Internet Explorer and the Silk Performer browser-driven load testing feature.

The certificate APIs only work with Microsoft Windows 7 and Internet Explorer 8 or 9.

Certificate handling for browser-based Web load testing works independently of certificate handling for protocol-based Web testing. This means that certificates need to be imported manually via Internet Explorer's **Internet Options** menu entry (or the management console `snap-in certmgr.msc`). If authentication works with Internet Explorer 8 it will also work for browser-based load testing.

1. When importing your certificate, disable strong private key protection:
   a) On the **Certificate Import** wizard **Password** page, uncheck the **Enable strong private key protection** checkbox.
2. Disable server certificate revocation:
   a) Open Internet Explorer's **Tools** menu and select **Internet Options**. The **Internet Options** dialog opens.
   b) Click the **Advanced** tab.
   c) Uncheck the **Check for server certificate revocation*** checkbox.
   d) Click **OK**.
3. Activate prompting of the client certificate selection dialog box:
   a) Open Internet Explorer's **Tools** menu and select **Internet Options**. The **Internet Options** dialog opens.
   b) Click the **Security** tab.
   c) Click **Custom Level...** The **Security Settings** page opens.
   d) Scroll down to **Don't prompt for client certificate selection when no certificates or only one certificate exists** and select the **Disable** option box.
   e) Click **OK**.
   f) Restart Internet Explorer.

# Removing Certificate Errors

During recording a Web page may appear with the message `There is a problem with this website's security certificate`. Additionally the `Continue to this website (not recommended)` link does not work. Certificate errors can occur due to multiple reasons and you must resolve any certificate errors before you can record a Web site browser-driven. For more information on certificate errors, visit *About certificate errors*.

One of the more common problems is an address mismatch. To disable address-mismatch warnings:

1. Open Internet Explorer's **Tools** menu and select **Internet Options**. The **Internet Options** dialog opens.
2. Click the **Advanced** tab.
3. Uncheck the **Warn about certificate address mismatch*** checkbox.
4. Click **OK**.
5. Restart Internet Explorer.

# Excluding URLs from AJAX Synchronization

To better facilitate the testing of AJAX-based Web applications, specific URLs can be excluded from browser synchronization.

To illustrate the value of this, imagine that an application displays server time by polling data from the server. This service requires a constant stream of traffic between the client and the server. This presents a challenge to AJAX synchronization because the application never goes into an idle state. By excluding this service from synchronization, other application processes that use different services can be accurately tested.

1. Right-click a profile in the **Project** menu tree and select **Edit Profile**. The **Profile - Simulation** window opens.
2. In the **Replay** group box, click the down arrow to scroll down. Click **Web (Browser Driven)**.
3. Select the **Replay** tab.
4. Enter URLs to be excluded into the **URLs to exclude from synchronization** text field.
5. Click **OK**.

**Note:** When URL exclusion is not feasible due to there being multiple processes running within a single service, you need to disable AJAX synchronization and switch to HTML mode.

# Index

## A



## B



## C



## D



## E



## H



## M

## N



## P



## S



## T



## U



## W