Silk Performer 19.5

Browser-Driven Web Load Testing Tutorial

MICRO FOCUS

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, Firefox, and Chrome) 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, Firefox, and Chrome.

With the browser-driven load testing support, Silk Performer strives to offer the same functionality and behavior across all supported browsers. This makes it possible to record your scripts in the Internet Explorer-based Browser Application and replay them with Firefox and Chrome.

In some rare cases however, this abstraction might not work as expected due to inherent differences among the browsers. For example: The Browser Application might generate a locator that cannot be resolved by Chrome or Firefox. For more information, see Limitations for Replay with Different Browsers.

Most browsers follow a frequent and silent update policy. For a load testing environment, this is not ideal, because it influences the consistency of the results. Furthermore, Silk Performer might not have been tested with the latest browser versions, especially if these have been published after the latest Silk Performer release. As a general rule: To avoid problems, stick to the tested browser versions listed in the section *Tested Software* in the release notes and install the latest hotfixes.

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: 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

Silk Performer recognizes the following dialog types:

JavaScript dialogs	supported in all browsers
Print dialogs	not supported
File Save & File Open dialogs	supported in Internet Explorer
Modal windows and modeless windows (HTML dialogs)*	supported in Internet Explorer

* HTML dialogs display as windows. Note that modal and modeless windows are a specific feature of Internet Explorer, hence they are only supported in Internet Explorer.

Native Replay

This topic applies to Internet Explorer only. For the other browsers, Silk Performer does not provide native replay capabilities.

The concept of *native replay* makes replaying a script more reliable. This is achieved by using Windows API-level events instead of JavaScript events for frequently used functions.

In the context of native replay we are using two special terms: *native replay* and *legacy input mode*. Note that native replay is the opposite of legacy input mode. If you turn on legacy input mode, native replay is automatically turned off, or the other way around. In this sense, we can also distinguish between *native functions* and *legacy functions*.

- Native functions are using Windows API-level events.
- Legacy functions are using JavaScript events.

The following functions can be called legacy functions, since they all have a native equivalent. If one of these legacy functions is detected during replay, then ...

- `BrowserClick` is replayed like `BrowserNativeClick`
- `BrowserDoubleClick` is replayed like `BrowserNativeDoubleClick`
- `BrowserSetText` is replayed like `BrowserTypeKeys`
- `BrowserSetPassword` is replayed like `BrowserTypeKeys`
- `BrowserMouseMove` is replayed like `BrowserNativeMouseMove`

If a native function cannot be performed instead of the legacy function, the legacy function is used as fallback and a warning message is logged. This can be the case if, for example, no mouse position can be determined to click the element.

Native replay is enabled by default. You can disable it in the Profile Settings: Click **Settings** > **Active Profile** > **Web (Browser Driven)**. On the **General** tab, enable **Legacy input mode**. Alternatively, you can add the following function to your script: `BrowserSetOption(BROWSER_OPT_LEGACY_INPUT_MODE, true)`

Legacy input mode is enabled by default for all project profiles created with Silk Performer 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.

- 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 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 - Enable switch to new tab when tabs are created	- Avoids unexpected dialog boxes - 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 - Avoids unexpected data input while typing
Programs	Manage add-ons	Only enable required add-ons	- Third-party add-ons may contain defects - Third-party add-ons may be incompatible
Advanced	Settings	- Disable **Automatically check for Internet Explorer updates** - Enable **Disable script debugging** (Internet Explorer) - Enable **Disable script debugging** (other) - Disable **Display notification about every script error** - Disable all **Warn...** settings	Avoids unexpected dialog boxes. ✎ **Note:** Depending on your browser version, not all settings may be available.

Running Multiple Virtual Users

When running multiple browser sessions in parallel, Silk Performer must make sure that these sessions operate independently from each other. Cache, cookie data base, and history in particular must not be shared across virtual users to simulate real user load. The separation of virtual users with regards to browsers is called *browser sandboxing*, meaning each virtual user runs its browser in its own environment.

To run browsers in sandboxed mode, Silk Performer uses different browser-dependent techniques.

Internet Explorer

For performance and resource reasons, Silk Performer does not use a full Internet Explorer browser instance to simulate a virtual user, but an Internet Explorer ActiveX control. The default behavior of the Internet Explorer control is to maintain a single cookie database, cache, and history across browser instances started by a particular Windows user. For load tests, Silk Performer reconfigures an Internet Explorer control to maintain one cookie database, cache, and history for each virtual user.

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

Chrome and Firefox

These browsers provide their own mechanism to create independent browser sessions. As a driver engin, Silk Performer uses the corresponding WebDriver interface and some custom enhancements to enable them to run in a load testing environment.

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).*

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.

Recording a Test Script

Recording browser-driven scripts is available for Internet Explorer only.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select Silk Performer **Browser Application** from the **Recording 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 (Internet Explorer).

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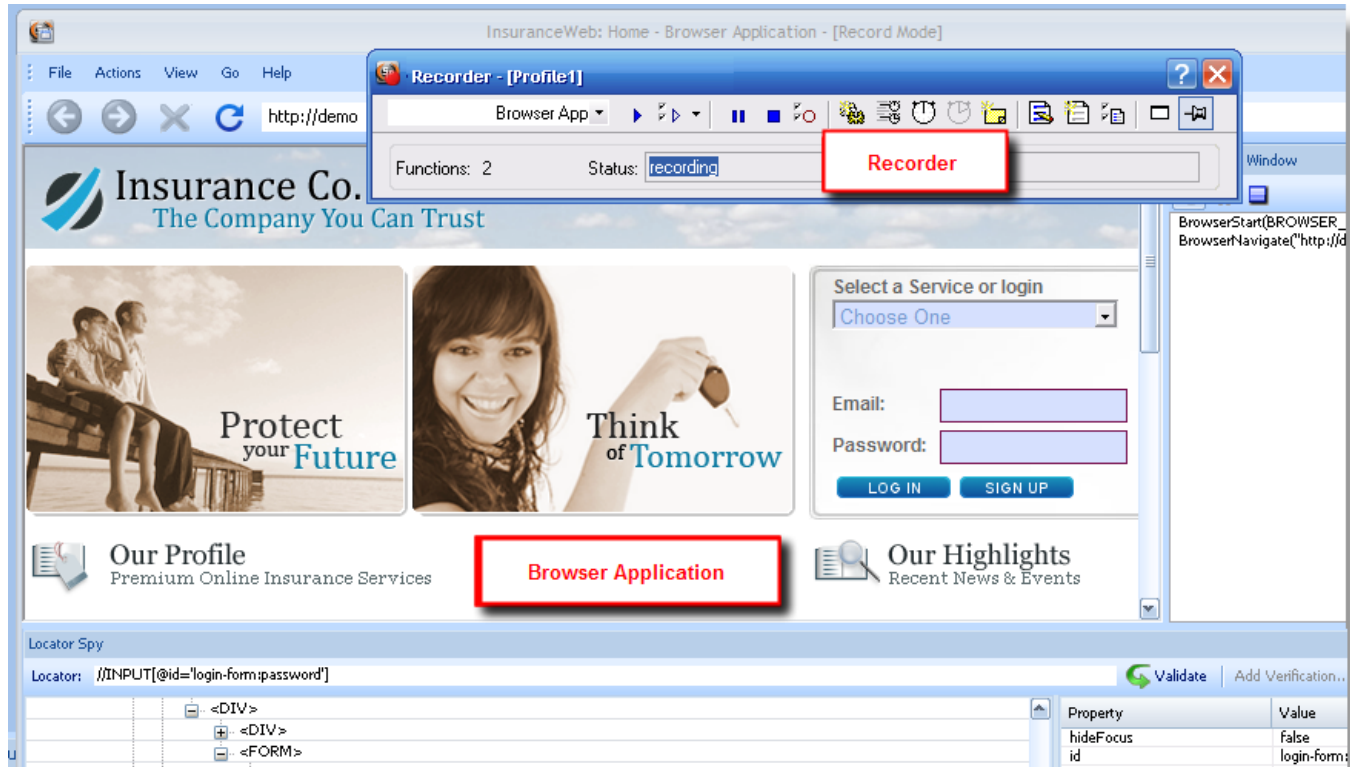
5. Using the Browser Application Recorder, interact with the sample application in the same way that you want your virtual users to act during the test (for example, click links, type data into fields, submit data, and open the pop-up window). Your actions will be captured and recorded by the Browser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

Browser Application and Locator Spy Usage

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

- **Browser Window** (the Internet Explorer control)
- **Locator Spy**
- **Record/Replay Window**

The following image shows the most important elements of the Browser Application.

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.

Record/Replay Window

This 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.

Browser Navigation Bar

The bar enables standard browser navigation.

Highlighted DOM Element

When you move your mouse over a web page, the DOM elements under the cursor are being highlighted in green. The green rectangles help you to get a feeling for the architecture of the web page and its DOM hierarchy.

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 displayed.
- The locator for the inspected DOM element is determined and displayed in the **Locator field**.

To search within the Locator Spy, press `Ctrl+F` on your keyboard. Alternatively, select **Actions** > **Find in DOM Tree**. You can search for strings within **Tags**, **Property names**, or **Property values**.

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 you select another DOM element in the DOM hierarchy tree, the locator for the DOM element is determined and displayed next to its HTML tag. The locator field is updated and the DOM element is highlighted in blue.

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 field. Press `Pause/Break` to refresh the hierarchy tree and to highlight the current DOM object. Locator strings in the DOM hierarchy tree are also removed as they are now invalid.

Locator Field

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

The locator field can be used to copy a locator string to another location, for example to a BDL script. Or you can use the field to manually edit locator strings. When you edit a locator, it is automatically being validated. If the locator is invalid, it is highlighted in red. If the locator is valid, it is highlighted in green.

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 way, you can conveniently exchange properties in the locator field.

Attributes of Inspected DOM Element

This is a list of attributes (name/value pairs) belonging to the currently inspected DOM element. If the current locator string does not fit your needs, you can manually build a specific 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.

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.
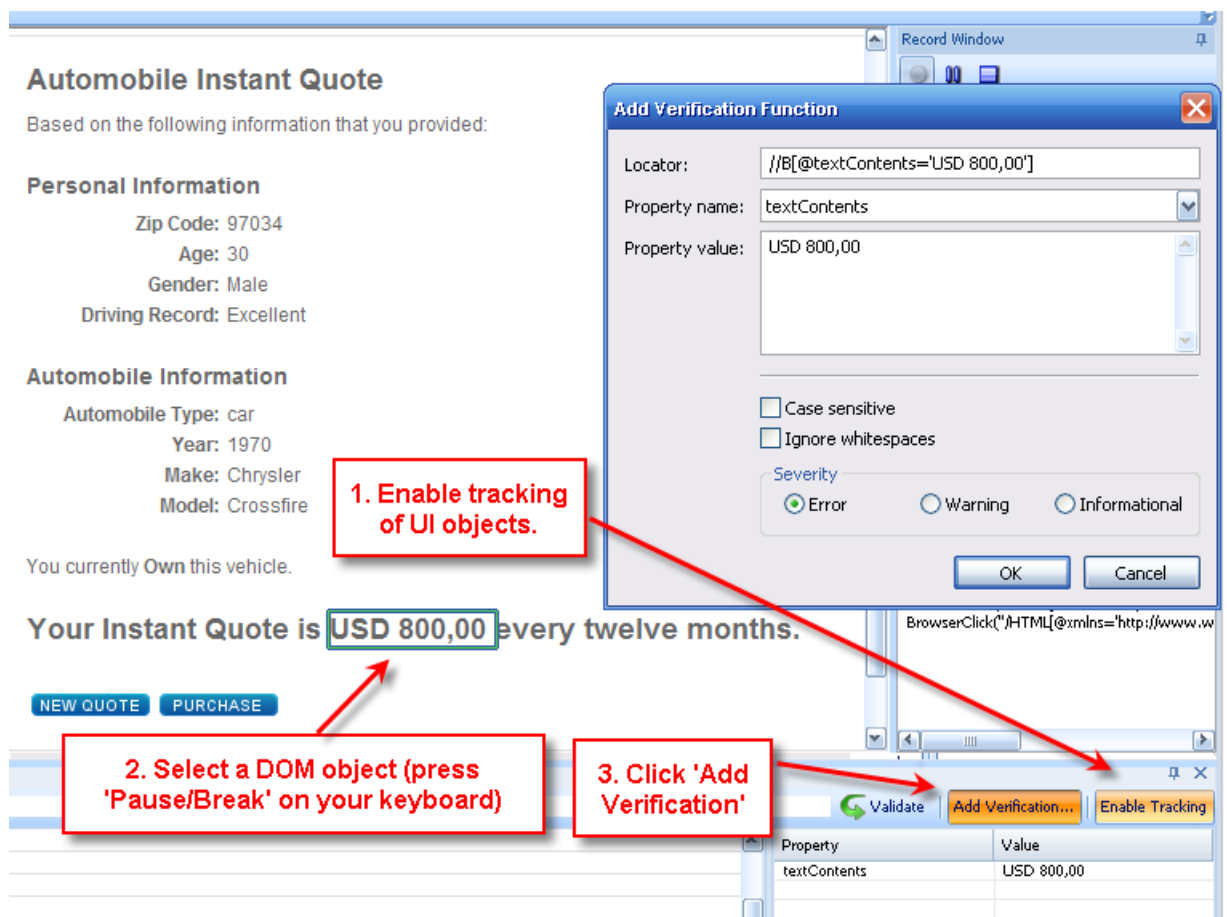2. Click **Add Verification**.

   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.

Including Elements in the TTI

1. Click **File** > **New Project** to create a new browser-driven project. If you want to use an existing project, click **Model Sript** on the workflow bar and skip to step 3.
2. Select **Web browser-driven (AJAX)** in the tree, enter a **Name** and **Description** and click **Next**.
3. Enter the **URL** of the application you want to record and click **Start recording**. The **Browser Application** launches.
4. Navigate through the application to record your actions. To tag an element as TTI-relevant, move your mouse over the element, press **Pause/Break** and click **Include in TTI**. You can include as many elements as you want. Silk Performer will add a `BrowserTtiIncludeElement()` function to the script.

   ⚠️ **Attention:** Make sure to not click the element before you include it. In such a case, the function `BrowserTtiIncludeElement()` will be tied to the click function, which might result in issues during playback.
5. Close the **Browser Application**, stop the recording, and save your 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) Select a **Browser** from the list: Internet Explorer, Mozilla Firefox, or Chrome.
   b) Enable the **Visible client** option so that the browser **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.

   c) Enable the **Step by step execution** option to run your script step by step. This option is available for Internet Explorer only.
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. This option is available for Internet Explorer only.

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 itself for script validation.

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

- Live application state is presented in the browser.
- Locator Spy functionality for advanced script modification and adaption (supported for Internet Explorer only).
- Scripts can be executed in step-by-step mode (supported for Internet Explorer only).
- 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
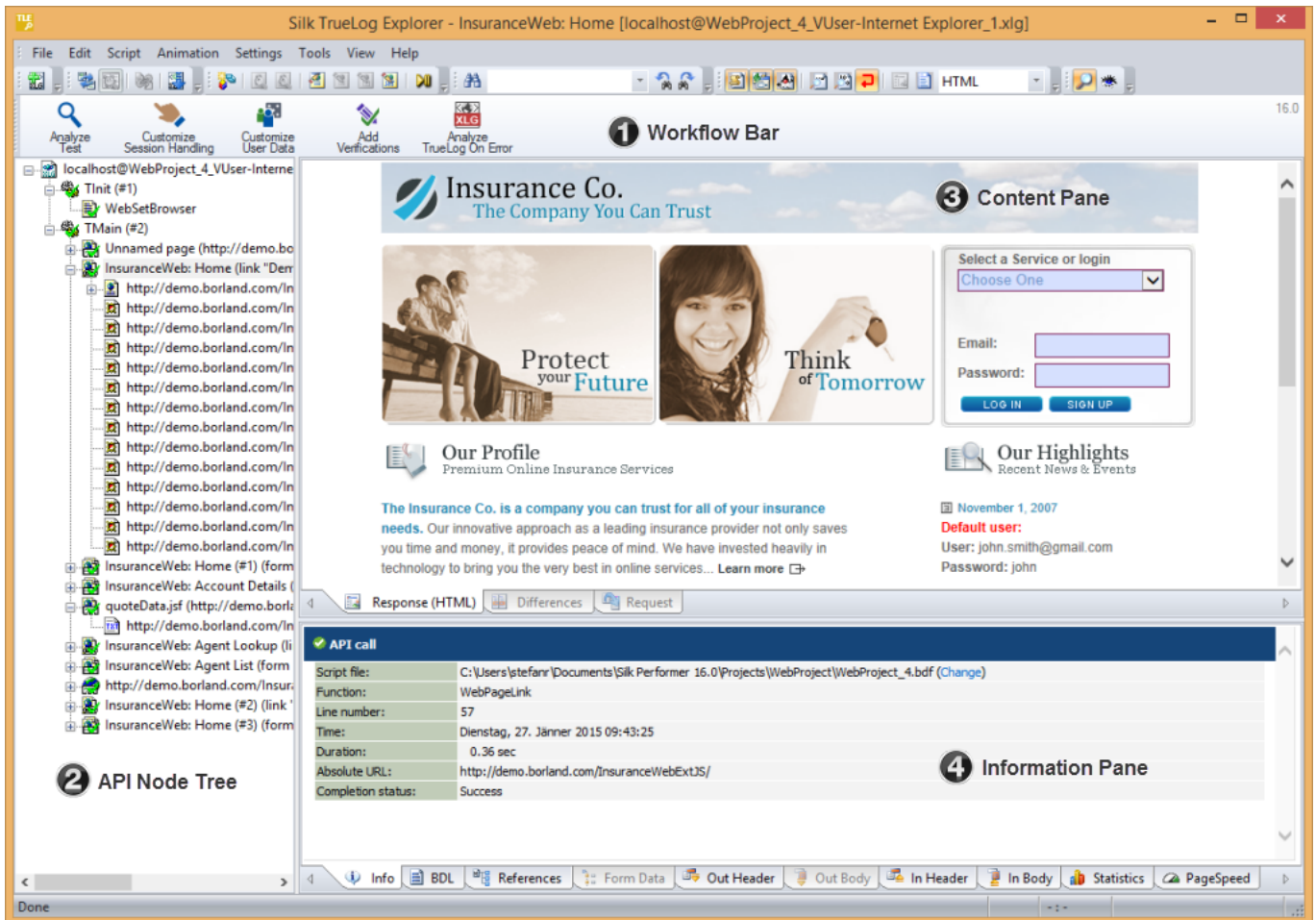- Comparing replay test runs with recorded test runs

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

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. Select the browser you want to use for this settings profile from the list.
5. 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. 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.
6. Select a replay compatibility to define how locators are generated. Setting this option ensures flawless replay of older scripts with newer versions of Silk Performer and therefor helps to avoid compatibility issues. You can also define the replay behavior for every single script by manually scripting the `BrowserReplayCompatibility` function.
7. Ensure that the **Legacy input mode** setting is disabled.
8. Click the **Synchronization** tab.
9. 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.

10. Click **OK**.

Advanced Concepts for Browser-driven Tests

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**.

Testing Websites That Use Non-system Codepage Characters

Silk Performer is a multibyte character set (MBCS) based application. When you use browser-driven load testing, you must set the correct system codepage. This ensures that the characters displayed on the website are processed correctly.

To enable browser-driven load testing of websites that use non-displayable characters, Silk Performer converts these characters. For example: The following string is converted to a set of numbers:

русский ⟶ [raw[440 443 441 441 43a 438 439]]

The numbers represent the Unicode value of each character in the hexadecimal format. When you replay a script, Silk Performer converts the string back and uses it while driving the browser.

> *Note:* Silk Performer applies the conversion only in Browser-Driven API calls. Do not use non-displayable characters in other API calls.

Time to Interact

### Testing AJAX websites is challenging

Measuring the user experience of AJAX websites with the timings the browser provides can be difficult. A user can consider a web page as ready, although the processing in the background is not yet completed. Also, the processing might be completed, but the web page is not yet ready for the user at that point in time. Essential page elements might be loaded asynchronously, that is after the *onLoad Function* phase and during the *Asynchronous Application Logic* phase. In such a case, the perceived loading time of a web page can differ considerably from the measured loading time. As a result, Silk Performer introduced the so-called *Time to Interact* (TTI).

The *Time to Interact*

In Silk Performer terminology, the *Time to Interact* is defined as the time from a user interaction (such as navigating to a URL or a click on a link) until all relevant elements a user requires to interact with the page are ready; even if the page has not yet completely loaded. Identifying the relevant elements of a page can not simply be automated, as it heavily depends on the use case and on the perspective which elements to consider *relevant*. For example: A web shop company can test their website from their own perspective and

from the perspective of their customers. From the company-perspective, the elements that contain the special offers might be considered relevant. But from the customer-perspective, just the search field might be considered relevant.

Therefore, the performance engineer has to tag all TTI-relevant elements during recording. The Recorder then generates a `BrowserTtiIncludeElement()` function for each of these tagged elements. During replay, Silk Performer measures how long it takes to load each TTI-relevant element and reports the maximum as the *Time to Interact*.
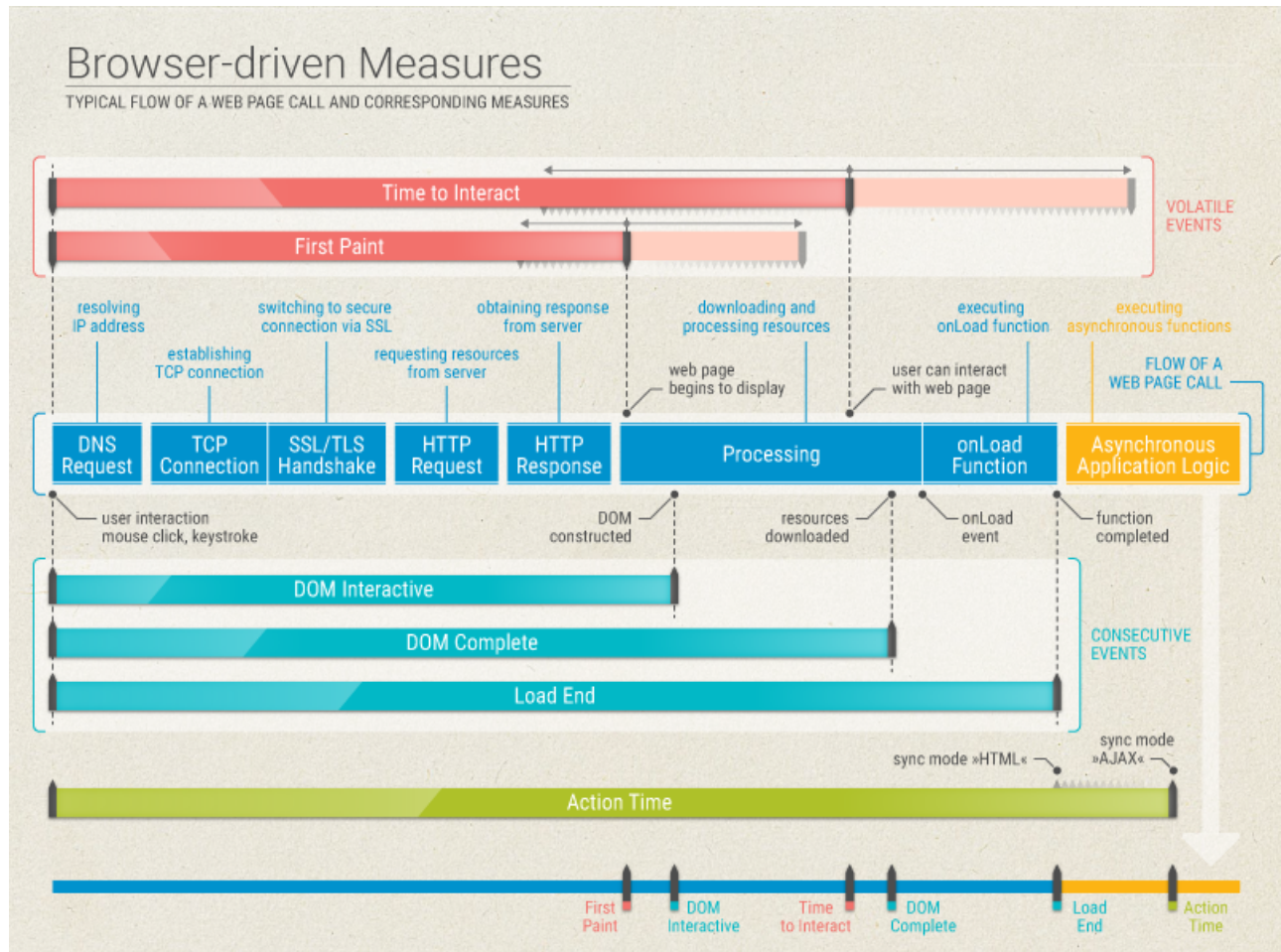
Browser-driven Measures

For browser-driven tests, Silk Performer provides the following measures:

Measure	Description
Dom interactive	Reflects the time span from the request until the browser has completed parsing all HTML elements and constructing the DOM.
Dom complete	Reflects the time span from the request until the browser has completed downloading and processing all resources (images, stylesheets, scripts, and so on).
Load End	Reflects the time span from the request until the browser has completed executing the onLoad function. As a final step in every page load process the browser sends an *onLoad* event, which triggers the onLoad function. Once the onLoad functions are executed, additional application logic might be executed.

Measure	Description
First paint	Reflects the time span from the request until the page begins to display. This measure is available for Internet Explorer only.
Time to interact	Reflects the time span from the request until all TTI-relevant elements are available on the page. At this point in time, the user can interact with the web page.
Action time	Reflects the time span from the request until the browser has completed downloading and processing all resources. The end of this time span varies, depending on the defined synchronization mode: If the synchronization mode `HTML` is defined, the action time ends when the onLoad function is completed. If the synchronization mode `AJAX` is defined, the action time ends during the *Asynchronous Application Logic* phase.

Note: **DOM interactive**, **DOM complete**, and **Load end** can be described as consecutive events. These events all end during the *Processing* phase. In contrast, **First paint** and **Time to interact** are completely website-dependent; they can end in the *HTTP Response* phase, in the *Processing*, and in the *Asynchronous Application Logic* phase.

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 10, 11.

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

The following applies only, when you use Internet Explorer.

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 work with Microsoft Windows 7 or later, Microsoft Windows Server 2008 R2 or later, and Internet Explorer 8 or later.

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

The following applies only, when you use Internet Explorer.

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 **Synchronization** 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.

Limitations for Replay with Different Browsers

Silk Performer uses Internet Explorer to record browser-driven scripts. To replay these scripts, you can use either Internet Explorer, Firefox, or Chrome.

Although these browsers basically resemble each other, they can behave quite differently in a variety of circumstances. This can also result in different behavior, when replaying the very same script with different browsers.

Below you can find a number of areas where you might encounter differences or issues. Note that many of the described issues are corner cases and rarely occur. Also note that the limitations heavily depend on your specific use case and on the application under test, including the underlying frameworks. Therefore, it is not possible to describe every potential limitation in every detail. If you encounter a specific limitation and need more information, contact SupportLine.

Mouse clicks

When replaying scripts, mouse clicks are processed differently in different browsers. Or in other words: The application under test receives different click events, depending on the used browser. For example: Replaying a double-click, raises a click and a double-click event in Internet Explorer and Firefox - this is the default behavior. But replaying a double-click in Chrome, raises just a double-click event.

Here is another example: When replaying a script in Chrome, the right-click event fails when the center of the clicked element is not visible. But in Internet Explorer and Firefox, the right-click event works flawlessly, even if the center of the clicked element is obscured.

Key strokes

Using key strokes within browsers also results in different behavior. For example: When the cursor is placed within a text field within Internet Explorer and you press the `Esc` key, everything you have typed so far is being removed. Pressing the `Esc` key in the other browsers has no effect. Another example is that Firefox and Chrome do not differentiate between the `Return` key and the `Enter` key on the numpad, while Internet Explorer does.

Locators

A browser might use an element attribute that the other browsers do not use. For example: Internet Explorer uses `spellcheck` as a generic attribute that exists for every element, while Chrome and Firefox only use the attribute when it is explicitly specified. Since Internet Explorer is used for recording the script, this results in an error when the script is replayed with Chrome or Firefox.

New windows

When your application under test uses a number of short-lived pop-up windows, this might result in unstable behavior. Here is a real-world example that might cause issues: You click a link that opens a pop-up window. This window immediately triggers another window to open, and then it closes again. When you use Firefox or Chrome, the window might be missed during replay. A possible workaround is to use the custom function `BrowserWaitForNewWindowWithLocator`, which is located in the `BrowserAPI.bdh`.

Invalid URLs

When you navigate to an invalid URL, Internet Explorer raises an error, while Chrome and Firefox do not. This can be problematic when monitoring a URL. The monitor will never raise an error, although the URL cannot be reached. For such a case, it can be useful to add a verification function to your script.

Measures

Due to the diverse replay technologies and the different architectures of the applications under test, some measures might be missing. For example: Some traffic might not be considered, because the application under test uses redirects or iframes.

Alert handling

When you use Firefox or Chrome to replay scripts, only JavaScript alerts and prompts can be handled. Native dialogs, like the file save or file open dialog, are not supported.

Compatibility mode

Scripts that are created using Silk Performer 18.5 or an earlier version, use a different *replay compatibility mode*. You can set the compatibility mode in the profile settings or by using the BDL function `BrowserReplayCompatibility`.

In the past, locators used to be tailored to Internet Explorer, because it was the only supported browser. Thus, the locators used in these older scripts might cause replay issues.

Index