SilkPerformer® 2010 R2
Browser-Driven Web Load Testing Tutorial
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Browser-Driven Web Load Testing Tutorial</strong></td>
<td>5</td>
</tr>
<tr>
<td>SilkPerformer Overview</td>
<td>5</td>
</tr>
<tr>
<td>Benefits of Using SilkPerformer</td>
<td>5</td>
</tr>
<tr>
<td>Reusing Projects With Test Manager</td>
<td>6</td>
</tr>
<tr>
<td>Support for Pop-Up Windows</td>
<td>6</td>
</tr>
<tr>
<td>Sample Web 2.0 Application</td>
<td>6</td>
</tr>
<tr>
<td>Pop-Up Window in the Sample Application</td>
<td>7</td>
</tr>
<tr>
<td><strong>Project Definition</strong></td>
<td>8</td>
</tr>
<tr>
<td>Defining a Browser-Driven Web Load Test Project</td>
<td>8</td>
</tr>
<tr>
<td>Creating a Test Script</td>
<td>9</td>
</tr>
<tr>
<td>Modeling a Test Script</td>
<td>9</td>
</tr>
<tr>
<td>Inserting a Verification Function</td>
<td>10</td>
</tr>
<tr>
<td>Try Script Runs</td>
<td>11</td>
</tr>
<tr>
<td>Trying Out Your Test Script</td>
<td>12</td>
</tr>
<tr>
<td>Common Replay Errors</td>
<td>12</td>
</tr>
<tr>
<td><strong>Analyzing Test Scripts</strong></td>
<td>13</td>
</tr>
<tr>
<td>Visual Analysis with TrueLog Explorer</td>
<td>13</td>
</tr>
<tr>
<td>Analyzing a Test Run</td>
<td>14</td>
</tr>
<tr>
<td>Viewing a Summary Report</td>
<td>15</td>
</tr>
<tr>
<td>Displaying a Virtual User Summary Report</td>
<td>15</td>
</tr>
<tr>
<td>Enabling Summary Reports</td>
<td>15</td>
</tr>
<tr>
<td>Finding Errors in a TrueLog</td>
<td>15</td>
</tr>
<tr>
<td>Viewing Page Statistics</td>
<td>16</td>
</tr>
<tr>
<td>Viewing an Overview Page</td>
<td>16</td>
</tr>
<tr>
<td>Comparing Record and Replay Truelogs</td>
<td>16</td>
</tr>
<tr>
<td><strong>Configuring Project Profile Settings</strong></td>
<td>17</td>
</tr>
<tr>
<td>Configuring Browser-Driven Recording Settings</td>
<td>17</td>
</tr>
<tr>
<td>Configuring Browser-Driven Replay Settings</td>
<td>18</td>
</tr>
<tr>
<td><strong>Technology Overview</strong></td>
<td>18</td>
</tr>
<tr>
<td>AJAX Overview</td>
<td>18</td>
</tr>
<tr>
<td>AJAX Processing Flow</td>
<td>20</td>
</tr>
<tr>
<td>Implications for Automation and Load Testing</td>
<td>21</td>
</tr>
<tr>
<td>Identifying DOM Elements Using XPath</td>
<td>22</td>
</tr>
<tr>
<td>Browser Application and Locator Spy Usage</td>
<td>23</td>
</tr>
<tr>
<td>Locator Verification in Browser Application</td>
<td>25</td>
</tr>
</tbody>
</table>
Browser-Driven Web Load Testing Tutorial

This tutorial will assist you in the process of using SilkPerformer 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 SilkPerformer's ease of use and exploit the leading-edge functionality that is embodied in e-business' load-testing tool of choice.

SilkPerformer Overview

In addition to facilitating testing of today's modern Web applications on the protocol level (HTTP), SilkPerformer 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 browsers 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), SilkPerformer supports the full range of Web applications that are developed for (and tested with) Internet Explorer.

Benefits of Using SilkPerformer

SilkPerformer is the industry's most powerful and easiest to use enterprise-class load and stress testing tool. Visual script generation techniques and the ability to test multiple application environments with thousands of concurrent virtual users allow you to thoroughly test your enterprise applications' reliability, performance, and scalability before they are deployed-regardless of their size and complexity. Powerful root-cause analysis tools and management reports help you isolate problems and make quick decisions, thereby minimizing test cycles and accelerating your time to market.

Following are some of the key advantages that SilkPerformer provides:

- *Ensure the scalability, performance, and reliability of your enterprise applications.* SilkPerformer ensures the quality of your enterprise applications by measuring their performance from the end-user perspective, as well as internally, in a variety of workload scenarios and dynamic load conditions.

- *Test remote components early in the development cycle.* Dramatically reduce the cost of defects in your multi-tier enterprise application by testing the functionality, interoperability, and performance of remote components early in the development cycle, even before client applications have been built. You can rapidly generate test drivers for Web services, .NET remoting objects, EJBs and Java RMI objects by exploring them via a point and click interface. Alternatively, you can reuse unit test drivers written by developers for concurrency tests or you can build new testcases directly in Java and .NET languages such as C# and VB.NET using the SilkPerformer Visual Studio .NET Add-On.

- *Pinpoint problems easily for quick resolution.* Unrivaled TrueLog™ technology for HTML, XML, SQL, Oracle Forms, Citrix, TCP/IP, and UDP-based protocol data provides full visual root-cause analysis from the end-user perspective. TrueLogs visually recreate the data that users input and receive during load tests. For HTML pages, this includes all embedded objects. This enables you to visually analyze the behavior of your application as errors occur during tests. In addition, detailed response timer statistics help you uncover the root causes of missed service level agreements before your application goes live.
Reusing Projects With Test Manager

SilkPerformer’s extended workflow simplifies and deepens integration with the Test Manager test-management solution. By clicking the Reuse Project button, test projects can be uploaded to and reused via Test Manager (for test automation). See Test Manager Help for details.

Support for Pop-Up Windows

SilkPerformer 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 SilkPerformer 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

SilkPerformer 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 SilkPerformer 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/index.jsf.

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.
Project Definition

The first step in creating a SilkPerformer load test is to define the basic settings for the load test project. The project is given a name, and optionally a brief description can be added. The type of application to be tested is specified from a range of choices that includes all of the major traffic available today on the Internet and on the Web, including the most important database and distributed applications.

The settings that are specified are associated with a particular load test project. It is easy to switch between different projects, to edit projects, and to save projects so that they can later be modified and reused.

A project contains all the resources needed to complete a load test. These include a workload, one or more profiles and load test scripts, a specific number of agent computers and information for server-side monitoring, and all the data files that are accessed from the script. Options for all of these resources are available directly from the project node in the Project menu tree.

Defining a Browser-Driven Web Load Test Project

1. Click Start here on the SilkPerformer workflow bar.
   The Outline Project dialog box displays.
2. Type a project name in the Project name field.
3. Type a description for the project in the Project description field.
4. Select Web browser-driven (AJAX) from the Application tree menu.
5. Click OK to create a project based on your settings.
Creating a Test Script

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

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

Modeling a Test Script

1. Click **Model Script** on the SilkPerformer workflow bar. The **Model Script** dialog box displays.
2. Select the **Record** option button.
3. Select SilkPerformer **Browser Application** from the **Select application profile** drop list.
4. In the **URL** text box, enter the URL that is to be recorded.
   
   **Note:** The InsuranceWeb sample Web 2.0 application is available at [http://demo.borland.com/InsuranceWebExtJS/](http://demo.borland.com/InsuranceWebExtJS/).
5. Click **OK**.
The SilkPerformer recorder opens in minimized form along with the SilkPerformer 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.

6. 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.

7. When you are finished, close the browser window and click Stop Recording. The Save As dialog box displays.

8. 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 active profile selected in the **Profile** drop list and the script you created selected in the **Script** drop 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.
      The **Animated** option is not available for this application type.
      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 enduring 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**: In some cases mouse movements are not recorded even when mouse-movement recording is enabled. AJAX applications often change the style of DOM elements on mouse-move events. If a locator relies on attributes that have changed due to a mouse-move event not being recorded, the resulting script will fail. The same applies to DOM elements that are attached to a document dynamically as a result of a mouse movement. You need to manually add BrowserMouseMove calls to your test script in such situations.

- **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 load 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:

- **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.
- **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 **Screen** pane displays a screenshot of the application’s state before the related API call was executed.
- **Information** view 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 SilkPerformer, choose **Results > Explorer 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 load 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 SilkPerformer 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 total page response times, document download times (including server busy times), and time elapsed for receipt of embedded objects.

Detailed Web page 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 Web page drill-down results include the following data for each page component:

- DNS lookup time
- Connection time
- Send request time
- Server busy time + response receipt 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

SilkPerformer 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.

 владельц: 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). The Recording tab displays.
4. 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.
5. 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.
6. The **Preferred DOM attribute names** option configures the name of the custom attributes that are recorded.

7. 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 (**Replay** category).

2. Scroll down to and select **Web (Browser Driven)**.
   The **Web (Browser Driven) Recording settings** tab opens.

3. Click the **Replay** tab.

4. 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).
   - 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.

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, SilkPerformer 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.

   **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. 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 (onreadystatechange)

**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.

SilkPerformer 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 SilkPerformer 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

SilkPerformer uses a subset of the XPath query language to identify DOM elements. For additional information about XPath, see [http://www.w3.org/TR/xpath20/](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 SilkPerformer.

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

The following table lists the XPath constructs that SilkPerformer does not support.
### Unsupported XPath Construct

<table>
<thead>
<tr>
<th>Unsupported XPath Construct</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparing two attributes to one another</td>
<td><code>a[@caption = @href]</code></td>
</tr>
<tr>
<td>Attribute names on the right side are not supported. Attribute names must be on the left side.</td>
<td><code>a['abc' = @caption]</code></td>
</tr>
<tr>
<td>Combining multiple XPath expressions with <code>and</code> or <code>or</code>.</td>
<td><code>a [@caption = 'abc'] or .//input</code></td>
</tr>
<tr>
<td>More than one set of attribute brackets</td>
<td><code>div[@class = 'abc'] [@id = '123']</code> (use <code>div[@caption = 'abc' and @id = '123']</code> instead)</td>
</tr>
<tr>
<td>More than one set of index brackets</td>
<td><code>a[1][2]</code></td>
</tr>
<tr>
<td>Wildcards in tag names or attribute names</td>
<td><code>*/@c?ption='abc'</code></td>
</tr>
<tr>
<td>Logical operators: <code>not</code>, <code>!=</code></td>
<td><code>a[@href!='someValue'], not[@href='someValue']</code></td>
</tr>
<tr>
<td>Any construct that does not explicitly specify a class or the class wildcard. For example, including a wildcard as part of a class name.</td>
<td><code>//@caption = 'abc'</code></td>
</tr>
</tbody>
</table>

### Browser Application and Locator Spy Usage

To enable convenient record/replay, SilkPerformer 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 SilkPerformer 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 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.

**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.
Web Browser Configuration Settings

Several browser settings are critical to maintaining stable test executions. Although SilkPerformer works without changing any settings, there are several reasons why you may want to change these browser settings in Windows 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
- Prevent browser malfunctions
  - Disable unnecessary third-party plug-ins

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

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

<table>
<thead>
<tr>
<th>Tab Name</th>
<th>Option</th>
<th>Configuration</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Home page</td>
<td>Set to <code>about:blank</code></td>
<td>Minimizes start-up time of new tabs.</td>
</tr>
<tr>
<td>General</td>
<td>Tabs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>· Disable warning for closing multiple tabs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>· Enable switch to new tab when tabs are created</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avoids unexpected dialog boxes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Links that open new tabs may not otherwise replay correctly</td>
</tr>
<tr>
<td>Privacy</td>
<td>Pop-up blocker</td>
<td>Disable pop-up blocker</td>
<td>Ensures that your Website can open new windows.</td>
</tr>
<tr>
<td>Content</td>
<td>Auto Complete</td>
<td>Turn off</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avoids unexpected dialog boxes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avoids unexpected data input while typing</td>
</tr>
<tr>
<td>Programs</td>
<td>Manage add-ons</td>
<td>Only enable required add-ons</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Third-party add-ons may contain defects</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Third-party add-ons may be incompatible</td>
</tr>
<tr>
<td>Advanced</td>
<td>Settings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>· Disable <strong>Automatically check for Internet Explorer updates</strong></td>
<td>Avoids unexpected dialog boxes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· Enable <strong>Disable script debugging</strong> (Internet Explorer)</td>
<td></td>
</tr>
</tbody>
</table>
## Running Multiple Virtual Users

Unlike other load testing tools, SilkPerformer 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, SilkPerformer 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.

## Troubleshooting Browser-Driven Load Testing Issues

**Note:** Browser-driven load testing is supported for Windows Internet Explorer versions 7 and 8.

### Starting Perfrun Process With System Account

Starting the perfrun process with an actual user account rather than the system account, which is the default, avoids the issue of recorded traffic differing from generated traffic. This issue is caused by the system account utilizing fewer or different HTTP headers than user accounts.

Windows Internet Explorer, which is utilized for browser-driven load testing requires user settings and behaves differently under the system account. Also, TrueLog screen-capture functionality cannot work when no user interface is available.

The required account setting is configured in System Configuration Manager on the **Applications** tab.

### 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 Windows Internet Explorer and the SilkPerformer browser-driven load testing feature.

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

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 Windows.
Internet Explorer's Internet Options menu entry (or the management console snap-in certmgr.msc). If authentication works with Windows 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 Windows 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 Windows Internet Explorer's Tools menu and select Internet Options. The Internet Options dialog opens.
   b) Click the Security tab.
   c) Click Custom Level...
   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 Windows 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, navigate to About certificate errors.

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

1. Open Windows 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 Windows 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

AJAX
analyzing a test run 14
analyzing test scripts 13
browser configuration 26
common replay errors 12
comparing record/replay TrueLogs 16
configuring recording settings 17
creating a test script 9
defining a project 8
displaying a summary report 15
enabling summary reports 15
Finding errors in a TrueLog 15
identifying DOM elements with XPATH 22
implications for automation and testing 21
Locator Spy 23
modeling a test script 9
overview 18
processing flow 20
project profile settings 17
replay settings 18
reusing projects with Test Manager 6
running multiple virtual users 27
sample Web 2.0 application 6
synchronization troubleshooting 27
technology overview 18
testing benefits 5
Try Script runs
  step-by-step replay 12
verification functions 10
viewing a summary report 15
viewing an overview page 16
viewing page statistics 16
visual analysis with TrueLog Explorer 13

identifying DOM elements with XPATH 22
implications for automation and testing 21
Locator Spy 23
modeling a test script 9
pop-up window in the sample Web 2.0 application 7
pop-up windows 6
prerequisites 27
project profile settings 17
replay settings 18
reusing projects with Test Manager 6
running multiple virtual users 27
sample Web 2.0 application 6
security certificates 27
testing benefits 5
troubleshooting 27
  Try Script runs
    step-by-step replay 12
verification functions 10
viewing a summary report 15
viewing an overview page 16
viewing page statistics 16
visual analysis with TrueLog Explorer 13

C

client certificates
  browser-driven Web testing 27

O

outlining
  projects 8

P

pop-up window support
  sample Web 2.0 application 6, 7
pop-up windows
  browser-driven Web testing 6
prerequisites
  browser-driven Web testing 27
projects
  outlining 8

S

security certificates
  browser-driven Web testing 27

T

troubleshooting
  browser-driven Web testing 27

browser-driven Web testing
  AJAX overview 18
  AJAX processing flow 20
  AJAX synchronization troubleshooting 27
  AJAX technology overview 18
  analyzing a test run 14
  analyzing test scripts 13
  browser configuration 26
  client certificates 27
  common replay errors 12
  comparing record/replay TrueLogs 16
  configuring recording settings 17
  creating a test script 9
  defining a project 8
  displaying a summary report 15
  enabling summary reports 15
  Finding errors in a TrueLog 15
Web 2.0 testing
  AJAX overview 18
  AJAX processing flow 20
  AJAX technology overview 18
  analyzing a test run 14
  analyzing test scripts 13
  browser configuration 26
  common replay errors 12
  comparing record/replay TrueLogs 16
  configuring recording settings 17
  creating a test script 9
  defining a project 8
  displaying a summary report 15
  enabling summary reports 15
  Finding errors in a TrueLog 15

identifying DOM elements with XPATH 22
implications for automation and testing 21
Locator Spy 23
modeling a test script 9
project profile settings 17
replay settings 18
reusing projects with Test Manager 6
running multiple virtual users 27
sample AJAX-based application 6, 7
testing benefits 5
  Try Script runs
    step-by-step replay 12
verification functions 10
viewing a summary report 15
viewing an overview page 16
viewing page statistics 16
visual analysis with TrueLog Explorer 13