MICRO FOCUS

Silk Performer 20.0

Help

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
http://www.microfocus.com

2019-04-16

Contents

Silk Performer Workbench 20.0

Welcome to Silk Performer Workbench 20.0.

Silk Performer is the enterprise-class solution for software-application performance, load, and stress testing. Silk Performer is used to assess the performance of Internet servers, database servers, distributed applications, and middleware, both before and after they are fully developed. Silk Performer helps you to quickly and cost-effectively produce reliable, high-quality application solutions.

What's New in Silk Performer 20.0

Silk Performer 20.0 introduces significant enhancements and changes.

New Editor

The code editor in Silk Performer 20.0 has been completely refreshed. It comes with a bulk of useful state-of-the-art IDE features, which greatly facilitates script development. The new code writing experience includes:

- line numbers
- rich find/replace functionality including find/replace in files
- an **Active Script** pane that updates in real-time
- auto indentation
- highlighting of same words
- code completion
- code collapsing
- code snippets insertion
- script error list pane and highlighting of syntax errors in the script
- inline variable declaration
- inline form view
- zooming
- split view
- column mode

The new editor makes you faster, thus you can keep concentrating on testing rather than writing test scripts.

Dynatrace Integrations

Dynatrace AppMon

Initially, the Silk Performer Dynatrace plugin was created by Dynatrace. Later on, the plugin was open-sourced and now with Silk Performer 20.0 the plugin is maintained by Micro Focus. The new plugin allows to disable request tagging for Try Script runs. Furthermore, the plugin enables users to configure the tag selection for the additional HTTP header. Linking from the HTML report to the Overall API breakdown in AppMon has been disabled due to a defect in the Dynatrace interface.

Dynatrace SaaS & Managed

The Dynatrace SaaS & Managed plugin not only offers similar functionality as the AppMon plugin but adds the possibility to send general information about the load test or error details to Dynatrace for further analysis. It automatically creates request attributes based on the selected tags and even creates request naming rules automatically within Dynatrace.

The Dynatrace SaaS & Managed plugin offers a set of tags to be sent within an additional HTTP header for each request.

CloudBurst Enhancements

Instance scheduling

With Silk Performer 20.0 you are now able to schedule cloud agent instances. By specifying the date and time you want to start testing, the requested number of agents will be ready to use. Thus, no time wasted with waiting for cloud instances to become ready.

Sorting options in Cloud Agent Manager

Cloud Agent Manager now allows to sort the agent list according to several columns. Additionally, the agents can be grouped by region.

Immediate feedback from cloud vendor

The Cloud Agent Manager now displays the current detailed state information in the tooltip of each agent status field.

Smart UI refreshing in Cloud Agent Manager

Collecting the current status from each cloud agent can be a lengthy, demanding process. Cloud Agent Manager now adapts the refresh interval depending on the state of each agent. If status updates are expected sooner the shorter is the refresh interval. During load tests when most agents are in use, status updates will occur less frequently.

Credit card payments method removed

The credit card payment option has been removed for various reasons. The preferred way to purchase Micro Focus credits is to contact a sales person.

Web on protocol-level Enhancements

WebSockets

WebSocket text messages can now also be used with binary data. Some technologies use text messages for sending binary data, rather than the binary message type.

Multipart form API

Silk Performer now offers additional BDL functions to handle multipart form post requests. For further detail see `WebAddMultiPart`, `WebAddMultiPartFromFile`, and `WebPostMultiPart`.

Other Enhancements

Silk Performer 20.0 provides a number of other enhancements:

- The new Micro Focus Community is fully integrated, including *Idea Exchange* as the replacement for *User Voice*. Relevant content has been migrated to the new platform, which can easily be accessed as usual from within the Silk Performer start page.
- Designing workloads can get quite complex and elaborate. The workload initialization dialog has been redesigned to better support performance engineers with defining workloads with multiple user types and consequently minimize the time required to setup complex workloads.

- The load test controller has been improved in stability and logging. The controller/agent communication is now more robust and fault tolerant under challenging network conditions. In addition, the Silk Performer SDK is now available in 64-bit, which allows integration in 64-bit native as well as Java 64-bit processes.

Technology Updates

OpenSSL

OpenSSL has been upgraded to version 1.1.1b. With this upgrade TLS 1.3 is the default security level used unless the server does not support it.

Other components

Several third-party components have been upgraded to their latest versions to remediate all known vulnerabilities and other defects.

Getting Started

Includes information about new functionality, an overview of load-test execution , and a performance and scalability matrix for workload configuration.

Introduction to Silk Performer

Silk Performer is the enterprise-class solution for software-application performance, load, and stress testing. Silk Performer is used to assess the performance of Internet servers, database servers, distributed applications, and middleware, both before and after they are fully developed. Silk Performer helps you to quickly and cost-effectively produce reliable, high-quality application solutions.

Silk Performer creates highly realistic and fully customizable load-tests . It accomplishes this task through the use of virtual users that automatically submit transactions to systems under test in the same manner as real users.

Using a minimum of hardware resources, you can generate tests that simulate hundreds or thousands of concurrent users. You can also use Silk Performer's powerful reporting tools both during and after load tests to analyze the performance of servers and to locate bottlenecks so you can maximize the potential of your system.

Questions That Silk Performer Can Help You Answer

- How many simultaneous users can my server support?
- What response times do my users experience during peak hours?
- Which hardware and software products do I need to ensure optimum performance from my server?
- Which components are the bottlenecks in my system?
- What is the performance impact on my system of employing security technology?
- Which areas of my application perform adequately, and which areas contain bottlenecks in the forms of business transactions, objects, and operations that can be evaluated?
- Which factors affect performance? What effects do they have? And at what point do such factors impact service levels?

Benefits of Using Silk Performer

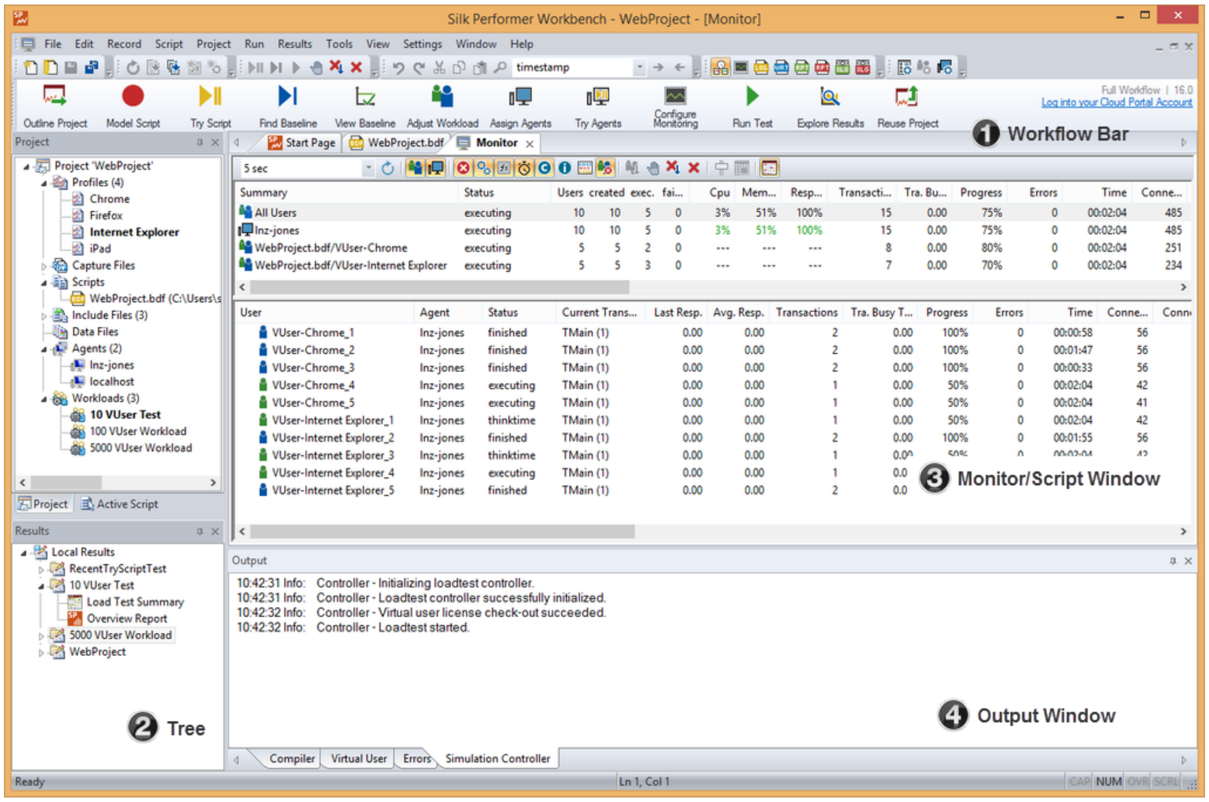
Silk Performer is the industry's most powerful and easiest to use enterprise-class load and stress testing tool. Visual script generation techniques and the ability to test multiple application environments with thousands of concurrent virtual users allow you to thoroughly test your enterprise applications' 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 Silk Performer provides:

- *Ensure the scalability, performance, and reliability of your enterprise applications*. Silk Performer 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 Silk Performer Visual Studio .NET Add-On.
- *Pinpoint problems easily for quick resolution*. Unrivaled TrueLog^{TM} 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.

Tour of the UI

Silk Performer's UI features four main areas:



Summary	Status	Users	created	exec.	fai...	Cpu	Mem...	Resp...	Transacti...	Tra. Bu...	Progress	Errors	Time	Conne...
All Users	executing	10	10	5	0	3%	51%	100%	15	0.00	75%	0	00:02:04	485
lnz-jones	executing	10	10	5	0	3%	51%	100%	15	0.00	75%	0	00:02:04	485
WebProject.bdf/VUser-Chrome	executing	5	5	2	0	---	---	---	8	0.00	80%	0	00:02:04	251
WebProject.bdf/VUser-Internet Explorer	executing	5	5	3	0	---	---	---	7	0.00	70%	0	00:02:04	234

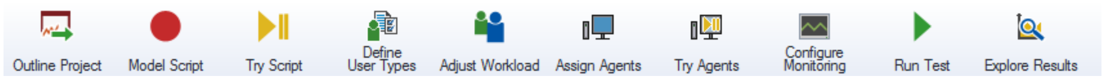
User	Agent	Status	Current Trans...	Last Resp.	Avg. Resp.	Transactions	Tra. Busy T...	Progress	Errors	Time	Conne...	Conn
VUser-Chrome_1	lnz-jones	finished	TMain (1)	0.00	0.00	2	0.00	100%	0	00:00:58	56	
VUser-Chrome_2	lnz-jones	finished	TMain (1)	0.00	0.00	2	0.00	100%	0	00:01:47	56	
VUser-Chrome_3	lnz-jones	finished	TMain (1)	0.00	0.00	2	0.00	100%	0	00:00:33	56	
VUser-Chrome_4	lnz-jones	executing	TMain (1)	0.00	0.00	1	0.00	50%	0	00:02:04	42	
VUser-Chrome_5	lnz-jones	executing	TMain (1)	0.00	0.00	1	0.00	50%	0	00:02:04	41	
VUser-Internet Explorer_1	lnz-jones	thinktime	TMain (1)	0.00	0.00	1	0.00	50%	0	00:02:04	42	
VUser-Internet Explorer_2	lnz-jones	finished	TMain (1)	0.00	0.00	2	0.00	100%	0	00:01:55	56	
VUser-Internet Explorer_3	lnz-jones	thinktime	TMain (1)	0.00	0.00	1	0.0[illegible]	50%	0	00:02:04	42	
VUser-Internet Explorer_4	lnz-jones	executing	TMain (1)	0.00	0.00	1	0.0					
VUser-Internet Explorer_5	lnz-jones	finished	TMain (1)	0.00	0.00	2	0.0					

```
10:42:31 Info:   Controller - Initializing loadtest controller.
10:42:31 Info:   Controller - Loadtest controller successfully initialized.
10:42:32 Info:   Controller - Virtual user license check-out succeeded.
10:42:32 Info:   Controller - Loadtest started.
```

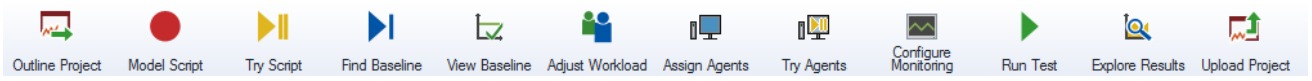
Workflow Bar

The workflow bar guides you sequentially through the steps involved in creating a typical load test. Each button on the following workflow bar represents a task, organized chronologically from left to right, that must be completed to successfully create, configure and run a load test, and to analyze load test results. There are three types of the workflow bar:

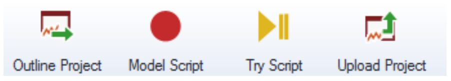
The simple workflow bar:



The full workflow bar:



The monitoring workflow bar:



Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Workflow bar buttons	
Outline Project	Give your project a name, description, and application-type profile.
Model Script	Create your test script. You can build your test script manually or have it recorded for you.
Try Script	Perform a trial run of your test script to make sure it runs as intended. If the trial run was successful, you can insert verification functions or parameterized input-data into your test script. Alternatively, you can edit your script so that it handles session information during replay.
Define User Types	A user type is a unique combination of a script, a user group, and a profile. Hence, by selecting a user type, you define which script shall be executed with what user group and what profile. User types are selected for a certain workload. You can use several workloads in your load test project and save them for further usage.
Find Baseline (only on the full workflow bar)	A baseline serves as a reference level for subsequent load tests. Silk Performer uses the results of a baseline test to calculate the number of concurrent users per user type.
View Baseline (only on the full workflow bar)	If you are happy with the results of a baseline test, you can set the baseline test as baseline. Then you can configure response time thresholds for the subsequent load tests.
Adjust Workload	Define the workload model that your test script will simulate and define the network bandwidth-type that the virtual users in your test will simulate.
Assign Agents	Configure the distribution of the virtual users in your load-testing environment. Assign VUsers to specific agents, agent clusters, or cloud agents, using wizards that calculate recommended capacities for you.
Try Agents	Perform a trial run of your script to make sure that it runs on the agents that you have configured for the load test.
Configure Monitoring	Define how Performance Explorer, the Silk Performer server monitoring tool, is to monitor local and remote servers involved in your test.

Workflow bar buttons	
Run Test	Deploy your test to the agent computers in your test environment. You can monitor the progress of the test in real time.
Explore Results	Analyze the results of the test to assess the performance of the application and server under test.
Upload Project (only on the full workflow bar)	You can reuse your project by uploading it to Silk Central or to Performance Manager. The project can then be directly executed through Silk Central or Performance Manager.

Project Menu Tree

Contains all of your project assets, including one or more profiles and test scripts, a workload, agent computers, and data files.

Monitor/Script Window

The **Monitor** page provides real-time information about the progress of your load test. The displayed columns depend on the application type of your project and can be customized by right-clicking either the **Summary** or the **User** list box on the **Monitor** page and choosing **Select Columns**. The selected columns are saved per project.

The **<script name>** page displays the script editor. Use the this page to view and edit your test script.

Output Window

Displays status messages, such as errors, warnings, and other information, when you perform the following actions:

- Compile a script – The **Compiler** page displays compiler messages, including errors and warnings.
- Run a test – The **Virtual User** page displays messages related to the actions of a virtual user in a test, such as transaction information and time measures. The **Simulation Controller** page displays information about the Silk Performer controller.

Managing Your Load Testing: Start to Finish Overview

This section offers a high-level overview of the steps involved in preparing for, running, and analyzing the results of a load test.

Each task outlined below correlates directly with a button on the Silk Performer workflow bar. The workflow bar guides you sequentially through the steps involved in creating and managing a typical load test.

Outlining Projects

When you create a Silk Performer load test you must 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.

Outlining a 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, choose the type of application that you want to use in your test.

   Note: If you are testing a Web application, choose the **Web business transaction (HTML/HTTP)** option to create simpler scripts while incorporating advanced functionality. Choose the **Web low level (HTTP)** option if you want to put the highest possible load on your application. Web low level scripts are more complex than Web business transaction scripts, which require more effort to customize. It is recommended that you use the **Web business transaction (HTML/HTTP)** instead of the **Web low level (HTTP)** scripts for browser based applications.
5. Click **Next**.

   Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.

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

Modeling Scripts

Before you can conduct a Silk Performer load test you need to create a test script that prescribes the actions of the simulated users run during the test. The script is written in Silk Performer's proprietary scripting language, the Benchmark Description Language (BDL).

Scripts can be created in different ways depending on the application type. The standard (and typically easiest) method for creating a test script is to use the Silk Performer Recorder to capture and record traffic that is representative of the type you need to simulate in your test. The Silk Performer Recorder automatically generates a BDL test script based on the recorded traffic.

Another method of creating a test script is to manually create a new script in BDL. A variant on the manual approach is to create a test script based on one of the sample BDL scripts that are provided by Silk Performer.

Recording a Test Script

If you are an experienced user, you may want to create a script on your own and write your code manually. You can start with a blank script or you can customize one of the preinstalled sample scripts.

However, you can also use the Silk Performer Recorder, which does the scripting and the recording work for you. Here are the basic steps you need to perform when you use the Silk Performer Recorder:

1. Open the project in which you want to work, or start a new project.
2. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
3. From the **Recording Profile** list, select the profile for the client application that you plan to test. If a profile has not yet been set up for the application you want to use, click **Settings** to the right of the list to set one up.
4. Depending on the project type, enter either the application's URL in the **URL** field or the location of the application in the **Command line** field.

5. Click **Start recording**. The Silk Performer Recorder dialog opens in minimized form, and the client application starts.
6. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.
7. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.
8. Insert transactions and timers into the test script during the recording phase. You can create as many transactions and timers as you want. To insert a transaction, click the **New Transaction** button. A transaction represents a piece of work that can be assigned to a virtual user.
9. In the ensuing dialog, enter a name for the transaction and click **OK**. The new transaction appears in the **Actions** log.
10. To insert a timer, click the **New Timer Session** button. A timer is a user-defined measurement period in a test. You should create timers for each component of a transaction for which you want to analyze performance. In the ensuing dialog, enter a name for the timer and click **OK**.
11. To end recording, click the **Stop Recording** button.
12. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.
13. Close the client application and close the Silk Performer Recorder.

Trying Out Scripts

Part of the process of conducting a Silk Performer load test is to perform a trial run of the test script that was created during script modeling.

Normally, this is traffic recorded by the Silk Performer Recorder during script modeling. For a trial run, or Try Script, of a test script, options are automatically selected so that you can see a live display of the actual data downloaded. Log files, TrueLog files, report files, output files, and error files are created so that you can later confirm that the script works properly. Only one user is run, and the stress test option is enabled so that there is no think time and no delay between transactions. However, at this stage, the measurements typical of a real load test are not performed.

Try Script Settings

For Try Script runs, the following options are automatically set to these specified values (see also "Replay Options"):

- The **Stress test** option is on, when think times are disabled.
- The **Stop virtual users after simulation time (Queuing Workload)** option is off.
- The **Virtual user log files (.log)** option is on.
- The **Virtual user output files (.wrt)** option is on.
- The **Virtual user report files (.rpt)** option is on.
- The **Virtual user report on error files (.rpt)** option is on.
- The **TrueLog files (.xlg)** option is on.
- The **TrueLog On Error files (.xlg)** option is off.
- The **Compute time series data (.tsd)** option is off.
- All logging detail options (**Results** > **Logging** and **Results** > **Internet Logging** page) are on.
- The **Enable all measure groups (TSD measure groups)** option is off.
- The **Bandwidth** option is set to `High Speed (unlimited)`.
- The **Downstream** option is set to `unlimited`.
- The **Upstream** option is set to `unlimited`
- The **Duplex** option is off.

Trying Out a Test Script

You must record or manually create a test script before you can run a Try Script.

1. Click the **Try Script** button on the Silk Performer Workflow bar. The **Workflow – Try Script** dialog appears.
2. Choose a script from the **Script** list box.
3. In the **Profile** list box, the currently active profile is selected (this is the default profile if you have not configured an alternate profile).
   a) To configure simulation settings for the selected profile, click **Settings** to the right of the list box.
   b) To configure project attributes, select the **Project Attributes** link.
4. In the **Usergroup** list of user groups and virtual users, select the user group from which you want to run a virtual user.

   Since this is a Try Script run, only one virtual user will be run.
5. To view the actual data that is downloaded from the Web server during the Try Script in real-time, select the **Animated Run with TrueLog Explorer** check box.

   If you are testing anything other than a Web application, you should disable this option.
6. Click **Run**. The Try Script begins.

All recorded think times are ignored during Try Script runs. The **Monitor** window opens, giving you detailed information about the progress of the Try Script run. If you have selected the **Animated** option, TrueLog Explorer opens. Here you can view the actual data that is downloaded during the Try Script run. If any errors occur during the Try Script run, TrueLog Explorer can help you to find the errors quickly and to customize session information. Once you have finished examining and customizing your script with TrueLog Explorer, your script should run without error.

Customizing Scripts

After you have recorded and tried out your script for the first time, you may need to customize it for two reasons:

- The script does not replay without errors.

  Session-specific data that was captured during recording is usually not valid during replay. This is because data like session IDs change every time a new communication session is started with the application under test. Session-specific data can be represented by variables and is, consequently, dynamically adjusted during each run. This leads to clean scripts that work flawlessly every time they are executed.
- You want to introduce more variety in virtual user behavior.

  The virtual users will behave identically during your load test, unless you introduce data or environmental variation, such as simulating different browsers, bandwidths, or paths through the application.

Therefore, it is recommended to customize your scripts before you start a load test. Customizing scripts offers the following advantages:

- Your virtual users will be diverse.

  For example, you can set them up to use different browsers and bandwidths from different locations to access the application under test.
- Your virtual users will behave more realistically and more natural.

  This can be achieved with randomized data. The virtual users will then, for example, input varying data in forms, like different names, addresses, and credit card numbers. They can vary the product names they search for, the products they order, and so on. This leads to accurate and meaningful result data.

Customizing scripts is typically done before you move on with the next step in the workflow bar.

Defining User Types

The next step in the process of conducting a Silk Performer load test is to define one or more user types for the active workload. A user type is a unique combination of a script, a user group, and a profile. By selecting a user type, you determine which script will be executed with which user group and profile.

To find out what your active workload is, expand the **Workloads** node in the **Project** tree. The active workload is shown in bold text.

Click **Define User Types** on the workflow bar to access all necessary settings.

Note: The **Define User Types** button on the workflow bar is only visible when the simple workflow bar is enabled. The workflow bar displays the simplified workflow by default. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar**, **Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Defining a User Type

To define user types for the active workload:

1. Click **Define User Types** on the workflow bar. The **Workflow - Define User Types** dialog box appears.
2. *Optional:* Narrow down the list of **Available User Types** by selecting entries from the **Script** and **Profile** lists.
3. *Optional:* Click **Add new** to create a new profile. If you select a profile from the list, you can click **Edit Profile** to adjust the settings of the profile. For example, you can define a certain browser and bandwidth for this profile. Click **Set as active** to make the selected profile the active one.
4. Select one or more user types in the **Available User Types** list and click the arrow buttons to assign the user types to the workload. You can also double-click user types to move them between the **Available User Types** list and the **User Types in Workload** list.
5. User types that are assigned to the workload are automatically selected for execution. If you want a certain user type to not be executed during the baseline run, deselect it.
6. Click **Next** to advance to the next step in the workflow.

Note: The **Define User Type** button is only visible when simple workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Finding Baselines

A baseline serves as a reference level for subsequent load tests. Silk Performer uses the results of a baseline test to calculate the number of concurrent users per user type. Additionally, you can set response time thresholds based on the results of a baseline.

At first, you need to run a baseline test. After a baseline test run is finished, you can view all measurements and details of the run on the **Baseline Test Summary** page and in the baseline report. If you are happy with the results, you can set the baseline test as your new baseline. You can view the results and values of your baseline at any time by clicking **View Baseline** on the workflow bar. This opens the **Baseline Summary** page.

Note: To work with baselines, you must enable the full workflow bar.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Finding a Baseline

Customize your test script by assigning a user profile to it before running a baseline test.

1. Click **Find Baseline** on the workflow bar. The **Workflow - Find Baseline** dialog box appears.
2. *Optional:* Narrow down the list of **Available User Types** by selecting entries from the **Script** and **Profile** lists.
3. *Optional:* Click **Add new** to create a new profile. If you select a profile from the list, you can click **Edit Profile** to adjust the settings of the profile. For example, you can define a certain browser and bandwidth for this profile. Click **Set as active** to make the selected profile the active one.
4. Select one or more user types in the **Available User Types** list and click the arrow buttons to assign the user types to the workload. You can also double-click user types to move them between the **Available User Types** list and the **User Types in Workload** list.
5. User types that are assigned to the workload are automatically selected for execution. If you want a certain user type to not be executed during the baseline run, deselect it.
6. Click **Run** to perform the baseline test.

Silk Performer runs a baseline test to calculate average measures against which future test runs are measured.

Note: The **Find Baseline** button is only visible when the full workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Viewing Baselines

The next step in the process of conducting a Silk Performer load test is to set the baseline test (see the previous step *Finding Baselines*) as your baseline. The baseline will serve as a reference level for subsequent load tests and should reflect the desired performance of the application under test. Once you have set a baseline test as your baseline, you can configure response time thresholds.

Note: To work with baselines, you must enable the full workflow bar.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Setting Response Time Thresholds

1. Click **View Baseline** on the workflow bar. The **Baseline Summary** page appears.
2. Click **Set response time thresholds** in the **Next Steps** area on the right side. The **Automatic Threshold Generation** dialog box opens.
3. Select the timers for which you want to set thresholds.
4. Specify the appropriate **Lower bound** and **Upper bound** multipliers for your load tests.
5. If the corresponding timers from the baseline test are `0`, specify minimum values.
6. If you want to raise an error or a warning message in case a threshold is exceeded, you can specify the severity of the raised message.
7. Click **OK**.

Note: The buttons **Find Baseline** and **View Baseline** are only visible when the full workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Adjusting Workload

Configuring workload is part of the process of conducting a load test. Silk Performer offers different workload models to be used as a basis for your load test. Before configuring workload, you must select the model that best fits your needs.

You can define more than one workload model in your load test project and save them for further usage, but only one workload model can be active at a time. The accepted baseline results are associated with a workload model. If you copy or rename a workload model, the accepted baseline results are copied or renamed accordingly.

Workload Models

Silk Performer provides the following workload models:

- **Increasing** – At the beginning of a load test, Silk Performer does not simulate the total number of users defined. Instead, it simulates only a specified part of them. Step by step, the workload increases until all the users specified in the user list are running.

  This workload model is especially useful when you want to find out at which load level your system crashes or does not respond within acceptable response times or error thresholds.
- **Steady State** – In this model, the same number of virtual users is employed throughout the test. Every virtual user executes the transactions defined in the load-testing script. When work is finished, the virtual user starts again with executing the transactions. No delay occurs between transactions, and the test completes when the specified simulation time is reached.

  This workload model is especially useful when you want to find out about the behavior of your tested system at a specific load level.
- **Dynamic** – You can manually change the number of virtual users in the test while it runs. After the maximum number of virtual users is set, the number can be increased or decreased within this limit at any time during the test. No simulation time is specified. You must finish the test manually.

  This workload model is especially useful when you want to experiment with different load levels and to have the control over the load level during a load test.
- **All Day** – This workload model allows you to define the distribution of your load in a flexible manner. You can assign different numbers of virtual users to any interval of the load test, and each user type can use a different load distribution. Therefore, you can design complex workload scenarios, such as workday workloads and weekly workloads. You can also adjust the load level during a load test for intervals that have not started executing.

  This workload model is especially useful when you want to model complex, long lasting workload scenarios in the most realistic way possible.
- **Queuing** – In this model, transactions are scheduled by following a prescribed arrival rate. This rate is a random value based on an average interval that is calculated from the simulation time and the number of transactions per user specified in `dcluser` section of your script. The load test finishes when all of the virtual users have completed their prescribed tasks.

  Note: With this model, tests may take longer than the specified simulation time because of the randomized arrival rates. For example, if you specify a simulation time of 3,000 seconds and want to execute 100 transactions, then you observe an average transaction arrival rate of 30 seconds.

  This workload model is especially useful when you want to simulate workloads that use queuing mechanisms to handle multiple concurrent requests. Typically, application servers like servlet engines or transaction servers, which are receiving their requests from Web servers and not from end users, can be accurately tested by using the queuing model.

- **Verification** – A verification test run is especially useful when combined with the extended verification functionality. This combination can then be used for regression tests of Web-based applications. A verification test performs a specified number of runs for a specific user type.

  This workload is especially useful when you want to automate the verification of Web applications and when you want to start the verification test from the command line interface.

Assigning Agents

In support of large-scale load testing, Silk Performer has consolidated all agent-to-workload assignment features within a single workflow step, available via the **Assign Agents** workflow bar button. Here you can configure the distribution of virtual users in your load testing environment and assign VUsers to specific agents, agent clusters, or cloud agents. Wizards are available to assist you in calculating recommended capacity for specific agents.

The **Assign Agents** workflow bar button helps you get started with the following tasks:

- Configuring individual agents and adding them to the workbench agent pool
- Assigning individual agents to your project
- Assigning clusters of agents with pre-defined capabilities to your project
- Configuring your project to use agents that run as virtual machines in the cloud.

Assigning Agents to Workload

This task can only be performed after you have configured workload for your project.

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Click the **Agent Assignment** tab.
3. Select the **Assignment type**:
   - **Static assignment to project agents** : Use this method to statically assign specific agent computers (rather than clusters of agents) to your project. No agent-availability check is performed with this method and agent locking is disabled. Select this method if you want to use Agents deployed in the cloud.
   - **Dynamic assignment to project agents** : With this method, workload is delivered using dynamic agent-assignment at execution time against the project's agents. Workload delivery is enhanced with agent-capability specifications to create optimized workload-to-agent assignments based on the capabilities of each agent. Agent locking at execution time is enabled with this method. Only responding agents that are not currently used by another controller are used with this method.
   - **Dynamic assignment to Silk Central agent cluster** : Silk Performer workload delivered by way of Silk Central can also use dynamic workload-to-agent assignment. Within Silk Performer you choose the name of the agent cluster (from the drop list) that should deliver your test's workload. Silk Central then provides the list of agent computers that are assigned to the cluster. Workload is then assigned to specific agents at the moment of execution based on the capabilities of the individual agents. After you connect to Silk Central, you are presented with the list of available agent clusters. In the right-most window, you can view the agents that are currently associated with the selected agent cluster.
4. Define the agents that are to deliver the workload for your test.
   - If you selected **Static assignment to project agents** , you can check the **Even user distribution** check box to distribute all existing user types evenly across all agents, depending on each agent's general replay capabilities. To use agents that run as virtual machines in the cloud, check the **Use cloud agents** check box. Click **Cloud Agent Manager** to manage your agents in the cloud.
   - If you selected **Dynamic assignment to project agents** , workload is delivered automatically using dynamic agent-assignment at execution time against the project's agents.
   - If you selected **Dynamic assignment to Silk Central agent cluster** , you are asked to log in to Silk Central. When you are logged in, you can select the available agent cluster.

   The **Agents** list box displays the available agents.

5. Check the **Agent resource utilization** check box to assign a maximum percentage of total virtual users that each agent can run based on the agent's replay capabilities.
6. Check the **Balance load across agents** check box to apportion workload across agents.
7. If you selected **Static assignment to project agents** , use the lower window of the **Agent Assignment** page to define workload assignments for user groups.

   Note: Available options vary depending on the selected workload model.
8. Click **User Distribution Overview** to view the assignment of virtual users to the agent computers that are currently available and then click **Close**.
9. Click **OK** to save your settings.

Workload will be assigned to agents based on the agent-assignment settings you have configured. If there are not enough agents with the required capabilities to deliver the required workload, you will be presented with an error message and details regarding the user types that did not receive an agent assignment.

Trying Out Agents

Before you start your load test, it can be useful to verify that the test script you created works on all agents you are planning to use for the load test.

Try Agents Settings

For Try Agents runs, the following options are automatically set to these specified values (see also "Replay Options"):

- The **Stress test** option is on, when think times are disabled.
- The **Stop virtual users after simulation time (Queuing Workload)** option is off.
- The **Virtual user log files (.log)** option is on.
- The **Virtual user output files (.wrt)** option is on.
- The **Virtual user report files (.rpt)** option is on.
- The **Virtual user report on error files (.rpt)** option is on.
- The **TrueLog files (.xlg)** option is off.
- The **TrueLog On Error files (.xlg)** option is on.
- The **Compute time series data (.tsd)** option is off.
- All logging detail options (**Results** > **Logging** and **Results** > **Internet Logging** page) are on.
- The **Enable all measure groups (TSD measure groups)** option is off.
- The **Bandwidth** option is set to `High Speed (unlimited)`.
- The **Downstream** option is set to `unlimited`.
- The **Upstream** option is set to `unlimited`
- The **Duplex** option is off.

Trying Out a Test Script On Agents

You must record or manually create a test script before you can try out your script on various agents.

1. Click the **Try Agents** button on the Silk Performer Workflow bar. The **Workflow - Try Agents** dialog box appears.
2. Select one or more user types from the list **User Types to execute**.

   Each user type will be executed on each selected agent. For example: If you select two user types and three agents, Silk Performer will start six test runs in total.
3. Select one or more agents from the list **Agents**.

   Note: You can select local agents and cloud agents, or more specifically: cloud regions. If you select a cloud region, all cloud agents from the respective region will be tested. It is not possible to test individual cloud agents. You can start cloud agents in the **Cloud Agent Manager**. Note that it can take some time until the cloud agents are ready to execute tests. A note beside the cloud regions tells you how many cloud agents are reachable and ready for a test execution. For example: **2/4 Agents ready** means that 4 agents were started in the **Cloud Agent Manager** and 2 of these are ready for execution.

4. *Optional:* Click **Enable think times** if you want to consider the think times in your script during the run. This option is disabled by default.

   Note: Usually, a Try Agent run is used to verify that a test script works correctly on various agents. For such a functional test, think times can be neglected, since they are a means to create a more realistic user behaviour and therefor a more realistic load. However, load issues can be neglected in a functional test.
5. Click **Run** to try out the script on the specified agents.
6. If you have selected cloud regions to be tested, the **Review Estimated Micro Focus Credits Consumption** dialog box displays. It gives you an estimation of how many Micro Focus Credits the runs will consume. Click **Accept and Run**.

The **Monitor** window opens, giving you detailed information about the progress of the Try Agents run. Once all runs are finished, the **Try Agents Summary** displays.

Configuring Monitoring

Before running a test you need to define how Performance Explorer, the Silk Performer server monitoring tool, is to monitor local and remote servers involved in your test. Server monitoring reveals, locates, and assists in resolving server bottlenecks, allowing you to examine the performance of operating systems and application servers.

Three monitoring options are available:

- **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
- **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.
- **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

Defining Monitoring Options

1. Click **Configure Monitoring** on the workflow bar. The **Workflow - Configure Monitoring** dialog box appears.
2. Select one of the following options and click **Next**:
   - **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
   - **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.

    - **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

   (for Default Monitoring and Custom Monitoring only) A confirmation dialog box will notify you if you have logging enabled. Logging may skew your test results.
3. Click **OK** to accept your logging settings or click **Cancel** to adjust your logging options (**Settings** > **Active Profile** > **Results** > **Logging**).
4. *(for Custom Monitoring only)* Performance Explorer starts and the **Data Source Wizard** opens. Complete the steps outlined in the wizard.
5. The **Workflow - Workload Configuration** dialog box appears. Click **OK** to accept your monitoring settings.

Running Tests

Real-time information regarding agent computers, virtual users, and transactions is displayed for you while load tests run. Real-time performance monitoring of the target server is presented in graphical format.

When a test is configured as a verification run, the following options are automatically set to the specified values:

- A **Baseline report** file is automatically created.
- The **Stop virtual users after simulation time (Queuing Workload)** option is disabled.
- The **Virtual user log files (.log) option** is disabled.
- The **Virtual user report files (.rpt)** option is enabled.
- The **Virtual user report on error files (.rpt)** option is enabled.
- The **Compute time series data (.tsd)** option is disabled.

Running a Load Test

Run a load test after you set up your testing environment and configure all test settings.

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Configure the workload that you plan to use in your load test.
3. *(Optional)* Click **Connect** on the **Workload Configuration** dialog box to initialize the agent connection without starting the test.
   a) Click **OK** on the **New Results Files Subdirectory** dialog box.
   b) Click **Start all** on the Silk Performer toolbar to manually start the test from monitor view.
4. Click **Run** to start the load test.
5. Click **OK** on the **New Results Files Subdirectory** dialog box.

   (Optional) To specify a name for the results subdirectory, uncheck the **Automatically generate unique subdirectory** check box and enter a name for the new subdirectory in the **Specify subdirectory for results files** text box.

Monitor test progress and server activity by viewing the Silk Performer tabular monitor view and the Performance Explorer graphical monitor view.

Exploring Results

Silk Performer offers several approaches to displaying, reporting, and analyzing test results. Defined measurements take place during tests and can be displayed in a variety of graphical and tabular forms. Options include the following:

- **Performance Explorer**: This is the primary tool used for viewing test results. A fully comprehensive array of graphic features displays the results in user-defined graphs with as many elements as are required. The results of different tests can be compared. There are extensive features for server

monitoring. A comprehensive HTML based overview report that combines user type statistics with time series test result information is also available.
- **TrueLog On Error:** Silk Performer provides full visual verification under load capabilities for various application types. It allows you to combine extensive content verification checks with full error drill-down analysis during load tests.
- **Virtual User Report files:** When enabled, these files contain the simulation results for each user. Details of the measurements for each individual user are presented in tabular form.
- **Virtual User Output files:** When enabled, these files contain the output of write statements used in test scripts.
- **Baseline Reports:** A detailed XML/XSL-based report that provides you with a summary table, transaction response-time details, timers for all accessed HTML pages, Web forms, and errors that occurred. This information is available for all user types involved in baseline tests.
- **Silk Central Reports:** Silk Performer projects can be integrated into Silk Central (Silk Central) test plans and directly executed from Silk Central. This allows for powerful test-result analysis and reporting. For detailed information on Silk Central reporting, refer to *Silk Central Help*.

Load Test Summary

When a load test run is complete, the **Load Test Summary** page appears. You can also open this page from the **Results** tree or by clicking **Explore Results** on the workflow bar. The **Load Test Summary** page contains:

- a **Quick Summary**, which gives you an overview about the test duration, users, agents, and errors.
- an **Available User Types** area, which you can use to drill down on the results for each user type. Select a user type from the list.

You can perform the following actions in the **Next Steps** and **Analyze Result Files** area on the right side:

- Click **Analyze load test** to view all detailed metrics in Performance Explorer.
- Click **Analyze errors** to view the errors in Performance Explorer (if any occurred). In the **Error Details** tab of Performance Explorer you can further drill into the errors. Double-click an error to open the TrueLog Explorer. This is only possible if you enabled **TrueLog On Error** on the **Workflow - Workload Configuration** dialog box before you started the load test.
- Click **Compare with baseline** to compare the results of this test with the results of the baseline. This button is only visible if you have already set a baseline.
- Click **Set as baseline** to make the test you have just run your baseline (your reference level) for the upcoming load tests.
- Click **Explore detailed report** to open a detailed load test report.
- Click **Open results folder** to view other result files like the virtual user report files or the virtual user output files for each virtual user.

Note: If you want to prevent the summary page to appear each time a test is complete, disable the **Show Summary Page** button in the toolbar of the **Monitor** page.

Upgrading to Silk Performer 20.0

Licensing: Silk Performer 20.0 requires a Silk Performer 20.0 license.

Project files: Project files can be upgraded from previous versions of Silk Performer by opening them in the current version. Downgrading from the current version to older Silk Performer versions has not been tested and is not supported.

Installation Information

Parallel installations: Silk Performer 20.0 (controller and agent) can be installed together with any older version of Silk Performer on the same machine without influencing each other.

Refer to the Silk Performer Installation Guide for installation instructions.

Evaluation: During the installation process, install one of the following versions:

- **Evaluation** – Installs an evaluation version of Silk Performer, which grants you full product functionality for 45 days. The usage is limited to 10 virtual users. To upgrade to a full version at a later point in time, contact your sales representative.
- **Licensed** – Installs an unrestricted version of Silk Performer, which requires a Silk Performer license.

Evaluation Version

If you have installed the evaluation version of Silk Performer 20.0, you can use the full product functionality for 30 days, limited to ten virtual users. The software stops operating when the evaluation period expires.

To ease you into the process of working with Silk Performer 20.0, it is recommended that you review *Web Load Testing Tutorial*. This PDF-based tutorial can be found in Silk Performer's documentation set.

Contact your sales representative to upgrade to the full version of Silk Performer 20.0.

Performance and Scalability Matrix

Maximum VUsers per Agent Computer

The following table shows the recommended maximum number of virtual users per agent computer based on the hardware resources of the agent computer and the type of tested web application.

Note: Testing of applications utilizing SSL reduces VUser capacity by 35%. Testing with TrueLog on Error enabled reduces VUser capacity by 30%. Testing a web application using low-level APIs (Web Low Level) increases VUser capacity by 20%.

System	OS	Web Business App	Web Business App with SSL (-35%)	TrueLog on Error (-30%)	TrueLog on Error with SSL	Web Low Level (+20%)
Intel Xeon Quadcore 3 GHz, 16 GB RAM	Windows Server 2008 R2	6500	4225	4550	2958	7800
Intel Core i7 Quadcore 2.8 GHz, 8 GB RAM	Windows 8.1	6500	4225	4550	2958	7800
Intel Core2 Duo 3 GHz, 4 GB RAM	Windows 7 Enterprise	4700	3055	3290	2139	5640

Note: Web Business App uses page-level APIs (HTML/HTTP). Web Low Level uses low-level APIs (HTTP).

VUsers per Cloud Agent

Note: The maximum number of VUsers per cloud-based agent is 1,000 regardless of the type of the tested application.

The following table shows the number of VUsers that can typically be supported per cloud-based agent, based on the type of the tested application.

Application Under Test	Supported VUsers per Cloud-Based Agent
Browser-Driven Load Testing	5
Java/.NET	250
Secure Web (SSL)	600

Application Under Test	Supported VUsers per Cloud-Based Agent
Web	800

Test Parameters

The numbers in the previous tables were determined by using a realistic workload with the following characteristics:

- The page structure of all requested web pages corresponds to popular public web pages.
- The average think time between page views for virtual users was 32 seconds.
- A single virtual user was emulated with four concurrent connections.

As a result, the number of simulated virtual users corresponds to a realistic number of concurrent users accessing a popular web site, providing the following information:

- Average hits per page: 39
- Average page size: 130 KB
- Average think time between pages: 32 sec

Agent Capacity for Web Protocol Virtual Users

Silk Performer determines the capacity of a machine by using a formula that takes the following parameters into consideration:

- Number of CPUs
- Number of cores per CPU
- CPU speed
- Memory size

The result is the number of virtual users that can execute a web protocol script on an agent machine with the corresponding parameters.

Agent Capacity for Other Virtual User Types

To determine the number of virtual users that can run on a particular agent machine, the number of web protocol virtual users is weighted by a particular factor that depends on the used technology.

Note: For some technologies there is a maximum number of virtual users defined per machine, merely due to OS limitations rather than CPU or memory constraints.

Memory Footprints by Application Type

Web Business Transaction (HTML/HTTP)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,935	1,935	1.89
25	10,323	413	0.40
50	20,400	408	0.40

Web Low Level (HTTP)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,044	2,044	2.00
25	12,136	485	0.47
50	22,732	455	0.44

Web Browser Driven (AJAX)

The memory footprint for browser-driven Web load tests depends on several factors. Because browser-driven load tests use Internet Explorer for replay, you should be aware of the following:

- The browser (Internet Explorer) uses memory. Depending on the version of Internet Explorer that you use, this memory footprint varies.
- Memory usage depends on how you have configured the caching settings in Internet Explorer.
- The application under test uses memory, which often increases as the VUser interacts with the application.

To get a basic understanding of how much memory a VUser requires when conducting a browser-driven Web load test, execute a try script run and monitor the memory usage (uncheck the **Visible Client** check box for exact results). Note that memory may increase while using the application under test though. It is recommended to script `BrowserStart` functions at certain points in your script to reset the browser state (cookies, cache and history).

Silverlight

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	11,040	11,040	10.78
25	13,640	546	0.53
50	16,152	323	0.32

Flash Remoting

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,912	1,912	1.87
25	12,776	511	0.50
50	20,896	418	0.41

WebDav (MS Outlook Web Access)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	3,588	3,588	3.50
25	17,724	709	0.69
50	33,364	667	0.65

Remedy Ars Web

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,288	2,288	2.23
25	9,688	388	0.38
50	17,340	347	0.34

Email (SMTP/POP)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,904	1,904	1.86
25	5,268	211	0.21
50	8,912	178	0.17

Directory Server (LDAP)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,920	2,920	2.85
25	not tested	N/A	N/A
50	not tested	N/A	N/A

FTP

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,420	2,420	2.36
25	5,352	214	0.21
50	8,348	167	0.16

TCP/IP based Application

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,912	1,912	1.87
25	5,624	225	0.22
50	9,500	190	0.19

Mixed Protocols

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	N/A	N/A	N/A
25	N/A	N/A	N/A
50	N/A	N/A	N/A

TN3270

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,224	2,224	2.17
25	7,424	297	0.29
50	13,652	273	0.27

TN5250

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,495	1,495	1.46
25	not tested	N/A	N/A
50	not tested	N/A	N/A

VT100/VT200

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,069	2,069	2.02
25	7,168	287	0.28
50	10,664	213	0.21

SAPGUI

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	30,656	30,656	29.94
25	110,180	4,407	4.30
50	194,532	3,891	3.80

SAP NetWeaver (Web)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,072	2,072	2.02
25	10,084	403	0.39
50	17,936	359	0.35

Peoplesoft

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,936	1,936	1.89
25	9,812	392	0.38
50	16,408	328	0.32

Oracle 11i

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	22,636	22,636	22.11
25	34,748	1,390	1.36
50	51,104	1,022	1.00

Siebel 7 Web Client (incl. IE Option Pack)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,256	2,256	2.20
25	11,532	461	0.45
50	20,576	412	0.40

Radius

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	16,104	16,104	15.73
25	21,284	851	0.83
50	26,268	525	0.51

XML/SOAP (recording Web Service client)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,676	2,676	2.61

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
25	8,460	338	0.33
50	16,436	329	0.32

.NET Framework using Visual Studio .Net Add-On

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	19,964	19,964	19.50
25	N/A	N/A	N/A
50	N/A	N/A	N/A

Java Framework

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	13,044	13,044	12.74
25	22,356	894	0.87
50	27,920	558	0.55

IIOP – Corba / EJB (RMI over IIOP)

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,736	2,736	2.67
25	31,696	1,268	1.24
50	61,960	1,239	1.21

Jacada

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	15,556	15,556	15.19
25	24,020	961	0.94
50	31,552	631	0.62

Citrix Server

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	3,900	3,900	3.81
25	9,000	360	0.35

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
50	11,500	230	0.22

Visual Basic

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,100	2,100	2.05
25	4,128	165	0.16
50	6,276	126	0.12

Oracle

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,732	1,732	1.69
25	3,696	148	0.14
50	5,792	116	0.11

ODBC

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	5,468	5,468	5.34
25	16,918	677	0.66
50	29,016	580	0.57

DB2 CLI

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	5,168	5,168	5.05
25	21,952	878	0.86
50	37,468	749	0.73

Tuxedo (all Feat.) + Clarify

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	2,400	2,400	2.34
25	Not tested	N/A	N/A

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
50	Not tested	N/A	N/A

Silk Test

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	13,000	13,000	12.7
25	N/A	N/A	N/A
50	N/A	N/A	N/A

Siebel 6/DB2

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	Not tested	N/A	N/A
25	Not tested	N/A	N/A
50	Not tested	N/A	N/A

Siebel 6/Oracle

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	9,428	9,428	9.21
25	66,540	2,662	2.60
50	125,720	2,514	2.46

Siebel 6/SQL Server

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	Not tested	N/A	N/A
25	Not tested	N/A	N/A
50	Not tested	N/A	N/A

GUI-level testing

Users	PerfRun (KB)	Per User (KB)	Per User (MB)
1	1,788	1,788	1.75
25	5,080	203	0.2
50	Not tested	Not tested	Not tested

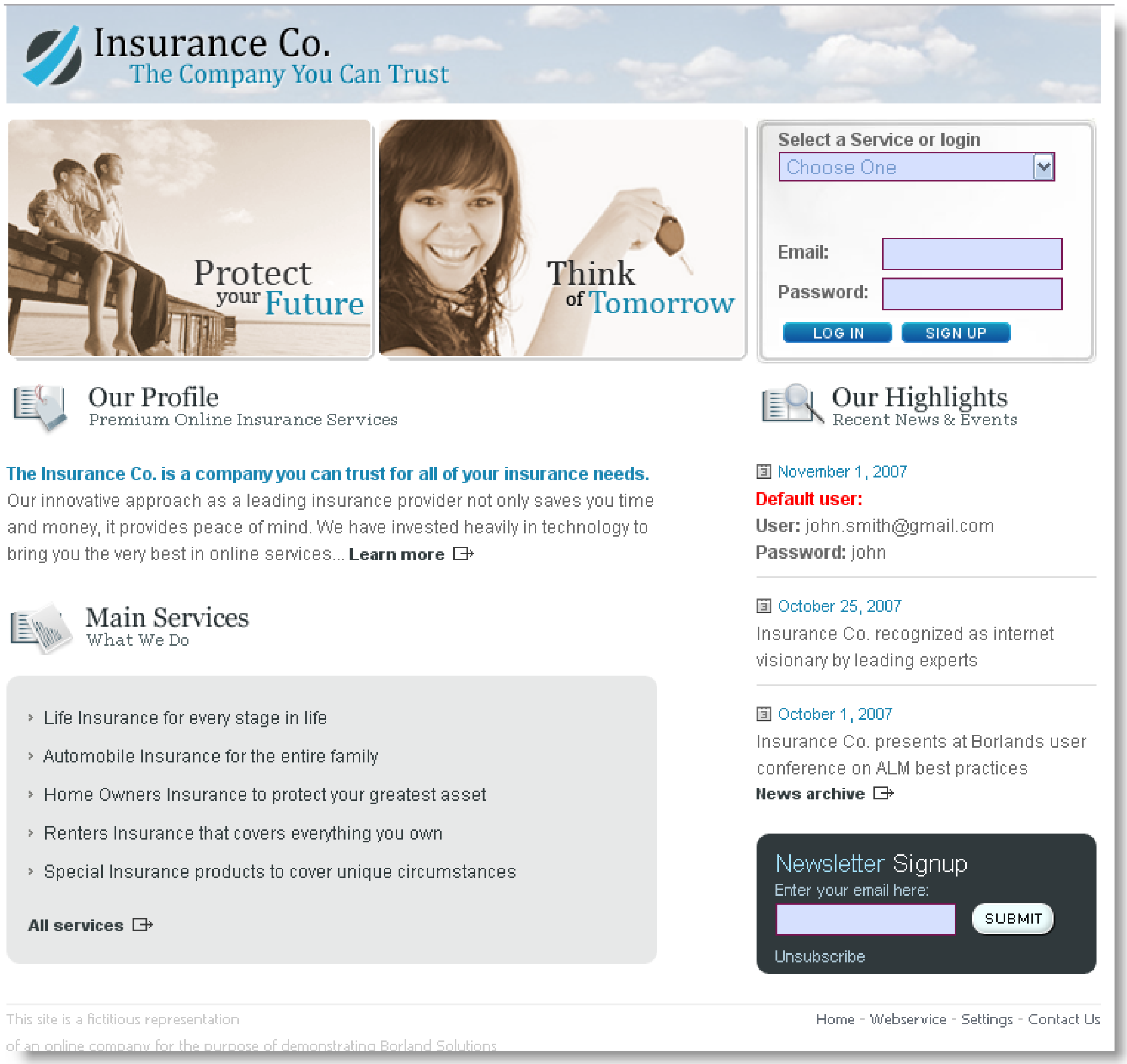
GUI-level tests require additional memory resources through the use of terminal services sessions and Silk Test, as follows:

- 6,300 KB per VUser for Windows Terminal Services
- 13,000 KB per VUser for SilkTest
- 1,092 KB per VUser for Silk Performer's Session Manager

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/*.


Insurance Co.
The Company You Can Trust

Protect your Future

Think of Tomorrow

Select a Service or login
Choose One
Email:
Password:
LOG IN SIGN UP

Our Profile
Premium Online Insurance Services

The Insurance Co. is a company you can trust for all of your insurance needs.
Our innovative approach as a leading insurance provider not only saves you time and money, it provides peace of mind. We have invested heavily in technology to bring you the very best in online services... **Learn more**

Main Services
What We Do

- Life Insurance for every stage in life
- Automobile Insurance for the entire family
- Home Owners Insurance to protect your greatest asset
- Renters Insurance that covers everything you own
- Special Insurance products to cover unique circumstances

All services

Our Highlights
Recent News & Events

November 1, 2007
Default user:
User: john.smith@gmail.com
Password: john

October 25, 2007
Insurance Co. recognized as internet visionary by leading experts

October 1, 2007
Insurance Co. presents at Borlands user conference on ALM best practices
News archive

Newsletter Signup
Enter your email here:
SUBMIT
Unsubscribe

This site is a fictitious representation
of an online company for the purpose of demonstrating Borland Solutions

Home - Webservice - Settings - Contact Us

Configuring Silk Performer

The Silk Performer configuration options provide the opportunity to set global Silk Performer settings that are not associated with a particular project.

Silk Performer Workbench: The basic characteristics of a Silk Performer load testing configuration are set here—those related to the agent computers used in tests, the directories where files used in and generated by tests are located, the layout of the text of test scripts, and how results information is generated.

Silk Performer Recorder: The basic settings used by Silk Performer's recording and script generating program, the Silk Performer Recorder, are defined here. These include the settings for capturing and recording the network traffic that will be modeled in your test scripts. Here, profiles are set up and managed for the client applications that are to be used during data capture and recording. Proxies can be added or removed, and their settings edited. Here you can set the port for the integrated Web server that is used by the Silk Performer Recorder, and enable recording to begin as soon as the client application starts.

Remote Agents: Silk Performer allows for remote agent computers to be located both in the LAN, out on the Internet, and even behind firewalls— connected through HTTP and SOCKS proxies. The communication between the remote agents and the controller can be configured to include secure, encrypted connections in a distributed secure environment.

Java Configuration: Java Framework projects and Java based APIs require a Java SDK Environment. The Java SDK version, its location, and parameters such as class path and optional Java Virtual Machine settings can be configured here.

System Settings

The basic characteristics of the Silk Performer testing configuration are set using the system settings. To access the system settings, click **Settings** > **System**.

Workbench Settings

To access the Workbench settings, click **Settings** > **System** > **Workbench**. The following tabs are available:

Tab	Description
Directories	Here you can specify where Silk Performer stores various types of files. Default directories are set up during installation; here you can specify alternative directories for projects, custom user data files, and custom include files.
Layout	Here you can customize the way the editor behaves when you manually insert text into a script file. Options can be set for formatting, printing, and how keywords are highlighted in the Silk Performer menu tree editor.
Control	Here you can set the options for the controller. Settings can be specified for the time interval within which a controller computer must communicate with an agent computer or a virtual user; if it fails, a timeout error will be reported. The frequency with which the controller will ping an agent computer until it receives a reply is set here, as can the number of virtual users per process and the number of transaction failures permitted for a virtual user before the user is terminated. **Note:** Virtual user settings are only stored on a per-project basis. You must have a project open to enable this setting. After a project is loaded, the options apply only to the specific project configured with that setting.

Tab	Description
Results	Use the **Time series data** option for merging time series data files (.tsd) on the agent computers where they were created and to ensure that only the merged files will be sent to the controller computer. This leads to quicker processing (because the data is processed by several computers) and less network traffic, which is especially useful for big and long-lasting load tests. The drawback, however, is that time series data files for individual virtual users are not available on the controller computer. **Note:** If you disable this option, you have to merge the time series data manually in Performance Explorer to get overall results for the load test.
Workspace	In this area, you can specify general Workbench settings including start-up settings, workflow settings, file handling, and the display of dialog boxes.
Licensing	In this area, you can specify settings for your license server and for the Silk Performer online licensing model.
Silk Central	In this area, you can set up a connection to Silk Central.
Source Control	In this area, you can specify general SCC connection parameters.

Configuring Directory Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Directories** tab.
3. Use the **File locations** area to specify the directories where various Silk Performer file types are stored. Default directories are set during installation.
4. In the **Projects** field, specify the directory where the project files are to be saved.
5. In the **Custom user data files** (`.pem`, `.rnd`, `.csv`, `.txt`, `.idl`) field, specify the directory where your self-created user data files (`.csv`), certificate files (`.pem`), random data files (`.rnd`), text files (`.txt`), and Interface Definition Language files (`.idl`) are located.

   In contrast to the specified **User data files** location, this location is used for your own user data files. As with Silk Performer pre-defined user data files, custom user data files in this directory can be shared across multiple projects.
6. In the **Custom include files** (`.bdh`) field, specify the directory where your self-created include files (`.bdh`) are located.

   In contrast to the specified **Include files** location, this location is used for your own include files. As with Silk Performer pre-defined include files, custom include files in this directory can be shared across multiple projects.

Configuring Layout Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Layout** tab.

   Use the **Editor** area to customize how the editor behaves when you manually insert text into a script file.
3. In the **Tab size** field, enter the number of space characters that equal one tab character.
4. Select the **Auto indent** option to automatically insert indents in your test script.

Each time you create a new line in a script file, the line is automatically indented the same number of spaces as the previous line.

5. Select the **Replace tabs with spaces** option to automatically insert four spaces each time a tab is used in a test script. This option makes it easier to view script files in a text editor.
6. Select **Enable CodeCompletion** to have functions, variables, and constants completed for you automatically when you enter text into BDL files.
7. Click the **Font & Colors** button to select a font and style for the Silk Performer script editor.

   You can select a different color to use for each type of content in a test script.

   The **Editor Font and Colors Settings** dialog opens.
8. From the **Text type** list, select the type of text for which you want to adjust the font, size, style, and color.

   Choose one of the following:

   - Normal Text - Identifiers and symbols.
   - Keywords - Predefined statements.
   - Custom Keywords - Words added to the keyword list by the user.
   - Numbers - Whole numbers and floating-point values.
   - Comments - Notes in explanation included in a script.
   - Strings - Sequence of characters delimited by " characters.
9. From the **Font name**, **Font size**, **Style**, and **Color** list boxes, select the format options you want to use for the currently selected text type.
10. Click **OK.**
11. You can add keywords to the **Custom keywords** list. These keywords will display in the script with the styles you have defined in the **Editor Font and Colors Settings**.

Configuring Print Layout Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Layout** tab.

   Use the **Printing** area to specify options for how script files are formatted when they are printed.
3. Select the **Line number** option to automatically number the lines in the printed script file.
4. Select the **Wrap marker** option to automatically print a marker at the end of each line that wraps to the next line.
5. Click the **Font** button to select a font and style for your printed test scripts. The **Printer Font and Colors Settings** dialog opens.
6. From the **Text type** list, select the type of text for which you want to adjust the font, size, and style.

   Choose one of the following:

   - Normal Text - Identifiers and symbols.
   - Keywords - Predefined statements.
   - Custom Keywords - Words added to the keyword list by the user.
   - Numbers - Whole numbers and floating-point values.
   - Comments - Notes in explanation included in a script.
   - Strings - Sequence of characters delimited by " characters.
7. From the **Font name**, **Font size**, and **Style** list boxes, select the format options you want to use for the currently selected text type.
8. Click **OK.**

Code Completion

Code completion makes it easier to work with BDL. It significantly reduces scripting errors and decreases the need to type text into BDL files by automatically completing functions, variables and constants. There are four code completion options:

- *Code list:* Shows a list box with all currently matching function names.
- *Code completion:* Completes the current function, variable, or constant name.
- *Parameter info:* Shows a tool tip with types and additional information for function parameters.
- *Quick info:* Shows information about functions in a tool tip.
- *Smart indention:* Positions the cursor based on the current logical context.

Code completion works with both Silk Performer-defined and user-defined BDL files.

To enable code completion, you must name and save new BDL files as either `.bdf` or `.bdh` files. Once a BDL file has been saved, code completion recognizes changes you make to it using the BDL Editor and includes all files you reference via BDL use statements.

Note: At all times, with all features of code completion, you are only able to receive information regarding functions, variables and constants that are defined in your local `.bdf` files, or in the included `.bdh` files of those files.

Code List

One of the key features of code completion is the code list mechanism, which offers a list of functions, variables, and constants. The code list includes either:

- The entire list of available functions, variables, and constants
- Or, a list of functions, variables, and constants that match the characters of the word currently selected in the script

The code list can be manually invoked using `Ctrl+Space`. The code list is invoked automatically when you type one of the following API prefixes:

Attribute	Iiop	DB_	Dotnet	Ora	Pdce
Java	Jolt	Measure	Odbc	Web	Xml
Set	Str	Tux	Get		

When the code list is invoked, a list control with the following behavior displays:

- If the cursor is not within a word or at the border of a word, the list control is scrolled to the top, and the first item receives focus, but it is not selected.
- If the cursor is within a word or at the border of a word, the list control is scrolled to the first (selected) item that matches the word, and the item is selected.
- By pressing the `Tab`, `Space`, or `Enter` key, the currently selected string is inserted into the view. If a string is not selected, the string with the focus is selected.
- Double-clicking an item in the list inserts the name into the view.

Code Completion

If a word that should be used with the code list feature is unique, the code list will not open when you invoke it via `Ctrl+Space`, however the full word will be inserted.

In this context, *unique* means that only one word in the list control matches the partial word in question.

Parameter Info

Parameter info is displayed when you place an opening bracket after a function name in a script. Parameter info can also be invoked manually by pressing `Ctrl+I` when the cursor is placed between the opening and closing brackets of a function call. Parameter info is displayed as a tool tip that remains visible until (1) the

`Esc` key is hit, (2) a closing bracket is inserted, (3) the arrow keys are used to move outside the brackets, or (4) input focus is set to another control.

Parameter info includes the following for functions:

- Function name
- Function parameters with names, types and modifiers (for example, `in`, `out`, and `allownull`)
- The current parameter is displayed with bold letters and, if available, an additional description
- Return value type and, if available, a description

Quick Info

Quick info is similar to parameter info except it shows information not only for functions, but also for variables and constants.

Quick info is presented in a form similar to a tool tip when you hold your cursor over a function, variable, or constant name. Quick info tool tips disappear when you move your cursor away.

For functions, available information is the same as with parameter info, with the exception that all parameters are shown with their descriptions rather than only the current parameter.

For variables and constants, only name, data type and optional descriptions are displayed.

Smart Indention

Smart indention is enabled when the `Enter` key is pressed within a BDL related view and a new line is to be inserted. Upon invocation, it places the cursor at a position that is calculated with respect to the current context (for example, indenting after inserting an initial keyword).

User-Defined Functions, Variables, and Constants

To receive code completion features for user-defined functions, variables, and constants, you must create a new file with the same name, in the same directory as your `.bdf` or `.bdh` file that defines the function, variable or constant you want to enhance, though with the extension `.bdd`.

A `.bdd` file has the following structure:

```
<CodeCompletionInfo>
  <Function id="FuncId">
    <Description>FuncDesc</Description>
    <ParamList>
      <Param no="1">
        <Name>ParamName</Name>
        <Description>ParamDesc</Description>
      </Param>
    </ParamList>
    <Returns>
      <Type>RetValTyp</Type>
      <Description>RetValDesc</Description>
    </Returns>
  </Function>
  <Global id="GlobalId">
    <Description>GlobalDesc</Description>
  </Global>
</CodeCompletionInfo>
```

This means that for each function you want to instrument with code completion information, you must insert a function node. For all others (variables and constants) you must insert a global node.

Code completion nodes

`FuncID`	Name of the function in your `.bdf` or `.bdh` file, for which code completion information should be provided.
`FuncDesc`	Textual description of the function.
`ParamName`	Parameter name
`ParamDesc`	Textual description of the parameter.
`RetValTyp`	Data type of the return value.
`RetValDesc`	Textual description of the return value.
`GlobalId`	Name of the global variable or constant.
`GlobalDesc`	Textual description of the variable or constant.

Configuring Control Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Control** tab.
3. The **Virtual users** area provides the following options:
   - Enable **Automatic calculation** to let Silk Performer calculate the number of virtual users per process.
   - Disable **Automatic calculation** to manually enter a value into the **Virtual users per process** field.
   - In the **Stop virtual users after __ transaction failures** field, enter the number of transaction failures after which virtual users should be stopped.

   Note: Virtual user settings are only stored on a per-project basis. You must have a project open to enable this setting. After a project is loaded, the options apply only to the specific project configured with that setting.
4. The **Remote agent communication** area provides the following options:
   - Select **LAN** to optimize the controller-agent communication for a local area network.

     This is especially useful if all of the agent computers you are going to use for your load tests are located in the same local area network and all are using a fast connection.
   - Select **WAN** to optimize the controller-agent communication for a wide area network.

     This is especially useful if you are going to use many agent computers that are located somewhere on the Internet.
   - Select **Custom** to specify your own parameters.

     Note: Use the **Custom** option only if you run into problems with LAN or WAN settings.
5. In the **Connect** field, enter the time, in which the controller computer should try to establish a connection to each remote agent computer.

   If the connection cannot be established within this time interval, Silk Performer reports an error.

   Note: It is not recommended to change this setting.
6. In the **Ping Interval** field, enter the time, within which the controller computer should attempt to retrieve load test status data from the agent computers. The controller computer requests this information from the agent computers whenever the contents of the **Monitor** window need to be updated. If the agent computer fails while receiving data during this time interval, Silk Performer will report a warning.

   Note: It is not recommended to change this setting.

7. In the **Command** field, enter the time in which the controller computer should try to send a command to a remote agent computer. If the command cannot be sent within this time interval, Silk Performer reports an error.

   Note: It is not recommended to change this setting.

8. Enable the compressing and caching options if necessary. For more information, read *Compressing Data Files* and *Caching Data Files*.

   - **Compress data files for LAN/WAN agents**
   - **Compress data files for cloud agents**
   - **Cache data files on agents**

Configuring Result Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Results** tab.
3. Select the **Merge time series data on agents** option in the **Time series data** area for time series data files (`.tsd`) to be merged in the agent computers where they were created and to ensure that only the merged files are sent to the controller computer.

   This is especially useful for quickly processing the results of long load tests as the calculations will then be distributed to various processors and less traffic will be sent over the network. The drawback, however, is that time series data files for individual virtual users are not available on the controller computer.

   Note: If you do not select this option, you will have to manually merge the time series data files in Performance Explorer to get overall results for the load test.
4. Enable **Limit messages per agent** and enter a value. If an agent exceeds this limit, Silk Performer reports a message with a customizable severity.

Configuring Workspace Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Workspace** tab.
3. Enable **Open the last opened project file on startup** to have the last project you worked on opened automatically each time you launch Silk Performer.
4. Check **Always enable all workflow buttons** to enable all buttons in the workflow bar.

   By default, workflow buttons are enabled in sequence based on your progress through the Silk Performer workflow.
5. Select a **Workflow bar** type:

   - **Full**: Displays the full workflow bar, including all workflow steps.
   - **Simple**: Displays the simple workflow bar. The simple workflow bypasses the baseline concept to make the load testing process quicker and easier.
   - **Monitoring**: Displays the monitoring workflow bar, including just the workflow steps necessary for monitoring purposes.
6. Check **Enable document file locking**.

   The document file locking mechanism allows multiple users (on Silk Performer Controller machines) to work simultaneously on the same document files (`.bdf` script files, `.bdh` include files, data files) without overwriting each other's files. Locked document files from other users can be used in read-only mode, but they can not be edited. This option is enabled by default.
7. Click **Show all dialog boxes** to reset any suppressed dialog boxes to their default active state.

   Certain dialog boxes include a **Do not show this dialog again** check box. Checking such a checkbox suppresses the dialog box from displaying until the **Show all dialog boxes** button is clicked.

Configuring Licensing Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Licensing** tab.
3. Click **Change license server configuration** to specify the settings for your Silk Meter licensing server.
4. Click **Install standalone license** to provide the location of your license file and import it.
5. Select one of the following **Licensing options**. These options allow you to use Silk Performer's online licensing model. For more information, read *Online Licensing*.

   - **Use on-premise licensing only**
   - **Use online licensing only**
   - **Use online licensing if required**

Recorder Settings

This topic explains the settings that are used by the Silk Performer recording and script generation program, the Silk Performer Recorder. These settings are used for capturing and recording the network traffic that is modeled in your test scripts. These settings comprise profiles that are applied to client applications during data capture and recording. Proxies can be added or removed, and their settings edited. You can set the port for the integrated Web server that is used by the Silk Performer Recorder, and enable recording to begin as soon as the client application is started. You can also configure recording rules that define how the Recorder captures network traffic.

Recording profiles

The Silk Performer Recorder uses recording profiles to determine what applications to record and how to record them. You can create new recording profiles and edit, remove, or copy existing ones. Within a recording profile you can specify the application that is to be recorded, a working directory, application arguments, additional executables to be recorded, protocol settings, and more.

Proxies

New proxies can be set up on your computer, and existing proxies can be edited or removed. A wide number of options can be specified for each proxy, including the protocol type, the listen port, details of the server to which the Silk Performer Recorder forwards intercepted traffic, and details concerning the client certificate that the Silk Performer Recorder presents to the Web server. When a SOCKS proxy is being set up, automatic protocol detection can be disabled for any port you want to specify. Whenever a client application sends SOCKS traffic to one of these specified ports, the Silk Performer Recorder records traffic at the TCP/IP level without trying to detect a familiar protocol first. A port range can be specified for which the recording of data by the Silk Performer Recorder will be suppressed.

Services

Options can be set for the Web server that is integrated into Silk Performer, and also for recording with the Silk Performer Recorder. The port that the integrated Web server listens to can be specified, and you can use this server to retrieve the root CA certificate that signed the Silk Performer Recorder server certificate. The root CA certificate may be downloaded from this Web site and installed into the Web browser to avoid security warnings by the browser.

The Silk Performer Recorder can be set to begin recording function calls automatically when an application is started from within the Silk Performer Recorder.

Recording Rules

The Silk Performer Recorder can be configured using recording rule files. Recording rule files are XML-based files that contain the rules by which the Silk Performer Recorder functions.

Recording rules allow you to configure the Recorder in a number of ways. Example use cases include session customizations, excluding JPG images from download while recording Web pages, and excluding Web pop-ups and ads that originate from a specific URL.

- TCP/IP: By providing protocol descriptions of proprietary TCP/IP based protocols.
- HTTP: By specifying the scenarios in which the Recorder should script parsing functions for dynamically changing values and generate replacements for those values.

Silk Performer enables you to create, edit, copy, and remove recording rules via the **Recording Rules** tab (**System Settings** > **Recorder** > **Recording Rules**).

Note: Creating recording rules based on existing templates, and editing individual rule properties as required, is the easiest method of configuring recording rules. Advanced users can manually script recording rules. Manually scripting recording rules requires extensive experience with Silk Performer and a thorough understanding of the involved protocols (TCP/IP, HTTP, and HTML). To learn about the structure and syntax of recording rules, and to see rule-file design examples, see the *Rule-based Recording* section.

You can create a new recording rule in TrueLog Explorer using values that you have specified in a parsing function. For example, creating a recording rule from a session-customization parsing function allows you to record an application while avoiding session customization issues entirely.

Note: All active recording rules are taken into account by the Recorder during recording. To avoid unwanted BDL scripting, ensure that only those rules that you require are active during recording. Recording rules are activated/deactivated via check boxes on the **Recording Rules** tab (**System Settings** > **Recorder** > **Recording Rules**).

Storing Recording Rules

The recording rules wizard saves recording rules in XRL format to `<public user documents>\Silk Performer 20.0\RecordingRules`. These rules can be shared between team members and can be added to any Silk Performer installation by copying the XRL file to any other `\RecordingRules` folder. Once in the folder, they are global in scope and are applied to all future projects.

You can control which recording rules are applied to subsequent recordings via **Settings** > **System** > **Recorder** > **Recording Rules**. To apply a rule, ensure the corresponding checkbox is checked. To have a rule ignored during recording uncheck the rule checkbox.

Adding Recording Profiles

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Click **Add** to add a new recording profile to the list.

   The **Recording Profile** dialog opens.
4. In the **Profile name** field, type a unique name for the recording profile.

   The profile name is required to identify the application in the Silk Performer Recorder window.
5. In the **Application path** field, type the name of the application's executable and the directory in which it resides.

   To locate the executable, click the browse icon.
6. In the **Working directory** field, type the working directory of the application.

   The working directory is the folder that contains the application or related files. Some applications need files located in other directories, so you might have to specify the folder in which these files reside.
7. In the **Application arguments** field, type any number of application parameters.

   Passing parameters to an application is useful, for example, when recording the function calls performed by a command-line tool without a graphical UI.

8. In the **Executables to be recorded** field, specify the executables that are to be recorded. This allows you to record processes that are different from the specified application's executable, which might be required if the application is hosted by another process that powers the communication process.
9. Enable **Record new and running instances (Secure Boot mode)** to record all instances of an application that are already running at the time the Recorder is launched, as well as all instances that are started with the Recorder or later on. You can enable this option in case the conventional recording mechanism does not work as expected for your application. This option is automatically enabled on machines that have the UEFI feature `secure boot` enabled.
10. In the **Protocol selection** area, check the check box that identifies the function library into which you want the Silk Performer Recorder to hook.

    Available options vary according to the selected application type, as follows:

    - **Web** – The Silk Performer Recorder hooks into a Web application, intercepts all function calls, and displays the results.

      When the **Web** option is selected, you can click **Web Settings** to open the **Web Settings** dialog box, which enables you to select the method that the Recorder uses to capture Web- and TCP/IP-based traffic.
    - **TCP/IP** – The Silk Performer Recorder hooks into a TCP/IP application, intercepts all function calls, and displays the results.
    - **IIOP** – The Silk Performer Recorder hooks into the IIOP function library of the client application, intercepts all the function calls, and displays the results.
    - **Citrix XenApp** – The Silk Performer Recorder hooks into a Citrix XenApp application, intercepts all function calls, and displays the results.
    - **ODBC** – The Silk Performer Recorder hooks into the ODBC function library of the client application, intercepts all function calls, and displays the results.
    - **DB2 CLI** – The Silk Performer Recorder hooks into the DB2 CLI function library of the client application, intercepts all function calls, and displays the results.
    - **Oracle OCI** – The Silk Performer Recorder hooks into the Oracle function library of the client application, intercept all functions calls, and displays the results.

      Click **OCI Settings** to open a dialog box that allows you to select the OCI client library to use for intercepting function calls. Select the appropriate library from the list box and click **OK**.
    - **Tuxedo** – The Silk Performer Recorder hooks into the TUXEDO function library of the client application, intercepts all function calls, and displays the results.
    - **SAPGUI** – The Silk Performer Recorder hooks into functions of SAP GUI interfaces, intercepts all the function calls, and displays the results.
11. Click **OK** to add the new profile to the list.

Removing a Recording Profile

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Select the recording profile you want to remove and click **Remove**.
4. Click **Yes** on the confirmation dialog.
5. Click **OK** to save your settings.

Editing or Copying Recording Profiles

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Select a recording profile.
4. Choose one of the following:

- Click **Edit** to adjust the settings of the selected recording profile.
- Click **Copy** to copy the selected recording profile.

The **Recording Profile** dialog appears. See "Adding Recording Profiles" for detailed information on the available settings.

5. Click **OK** to save your settings.

Recording Methods for Recording Profiles

You can set one of the following recording methods for each of your recording profiles. To view your recording profiles, click **Settings** > **System** > **Recorder**.

record new instances	When this method is set, the Recorder records all instances of an application that are started with the Recorder or afterwards. Instances that are already running at the time the Recorder is launched will not be recorded. To set this method, select a recording profile, click **Edit**, and make sure that **Record new and running instances (Secure Boot mode)** is disabled. This option is automatically enabled on machines that have the UEFI feature `secure boot` enabled.
record new and running instances	When this method is set, the Recorder records all instances of an application that are already running at the time the Recorder is launched, as well as all instances that are started with the Recorder or afterwards. To set this method, select a recording profile, click **Edit**, and enable **Record new and running instances (Secure Boot mode)**.
record through system proxy	When this method is set, the Recorder records through the Windows system proxy. **Note:** In this case, the Recorder captures the traffic of all applications that use the system proxy. You can exclude unwanted traffic from your script by using the filters and script generation settings on the **Capture File** page. To set this method, select a recording profile, click **Edit**, click **Web Settings**, select **Proxy**, and select **Automatic browser configuration**.
record through application proxy	Some applications (including Firefox) allow you to use an application-specific proxy and therefore ignore the system proxy settings. When this method is set, the Recorder records through the custom proxy of the respective application. To set this method, select a recording profile, click **Edit**, click **Web Settings**, select **Proxy**, and select **Manual browser configuration**. In the application you want to record, set the proxy to `localhost` and the HTTP proxy port configured in the Recorder (the default is `8080`).

The **Recording Profiles** page lists all recording profiles you have specified manually alongside a number of default recording profiles. When a recording profile is checked in this list, it is marked as `active`. All active recording profiles can be selected from the **Recording Profiles** list on the **Model Script** dialog.

You can select which of your recording profiles the Recorder uses for recording:

- **Profiles started from the Model Script dialog or recorder**: When this option is selected, only the recording profile that is selected on the **Model Script** dialog is used, once you start the recording. However, when you select a recording profile directly within the Recorder and click the **Start Application** icon, this recording profile is used as well, since it is initiated explicitly.
- **All active profiles**: When this option is selected, the Recorder uses all recording profiles that are `active` (checked), once you start the recording.

  Note: When you have selected this option and start your recording from the **Model Script** dialog, you still have to select a single **Recording profile**. However, the Recorder will use all active recording profiles during recording.

Tip: The Recorder shows you which recording profiles are in use by displaying the icons of the recorded applications on the bottom right.

Specifying the Recorder Method for Capturing Web and TCP/IP-Based Traffic

Before you can perform this task, you must add a recording profile.

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Select a recording profile.
4. Click the **Edit** button. The **Recording Profile** dialog appears.
5. In the **Protocol selection** area, select the **Web** option to have the Silk Performer Recorder hook into a Web application, intercept all function calls, and display results.

   The options available here vary based on the selected application type.
6. Click the **Web Settings** button to open the **Web Settings** dialog, which enables you to select the method that the Recorder is to use to capture Web and TCP/IP-based traffic.
   - Select the **WinSock** option to have the Silk Performer Recorder hook into the WinSock function library of the client application, wrap all traffic the application produces in the SOCKS protocol, and redirect the traffic to a Recorder SOCKS proxy.
   - Select the **Proxy** option to have the Silk Performer Recorder redirect all traffic from the selected browser to the Recorder proxies.
7. Click **OK**.
8. You can also enable any available Java APIs by checking the corresponding check boxes in the field on the right of the dialog. The following Java APIs are available for selection:
   - Oracle Forms 6i on HTTP/HTTPS
   - Oracle Forms 6i on Socket
   - Oracle Forms 9i
   - Product Manager Sample Extensions
   - Borland Enterprise Server App. Client
   - Jacada
   - Java EJB (JBoss)
   - Java EJB (Other)
   - Java EJB (WebLogic)
   - Java EJB (WebSphere)
   - Java JNDI (Naming & Directory Interf.)
   - Java RMI (Custom Client-Server)
   - Jolt
   - Oracle Applications 11i
   - Oracle Applications 12i on Socket
   - Oracle Forms 10g
   - Oracle Forms 11g

- Oracle Forms 12c

9. When a Java API is enabled, click the **Java Settings** button to open a dialog that enables you to specify Java recording settings, such as the hooked Java Virtual Machine library or other settings.

   - Select the **Java Virtual machine hooking** option to automatically hook into the Java VM. If this option is enabled, select the dynamic link library from the **JVM DLL** list box that you want the Silk Performer Recorder to hook into for recording traffic.
   - Select the **Network only** option to only hook class and jar files that are downloaded via HTTP.
   - Select the **Manual** option if you have manually prepared a startup script for recording your Java application.

10. Close all open dialogs by clicking the **OK** button to save your settings.

Configuring Proxy Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Click the **Proxies** tab.
4. Select one of the following options:

   - Click **Add** to set up a new proxy on the controller computer. You can set up proxies for recording a number of types of traffic, and you can have a number of proxies active at the same time.
   - Click **Edit** to edit a selected proxy.
   - Click **Remove** to remove a selected proxy.

   The **Proxy Settings** dialog appears.
5. From the **Protocol** list box, select a protocol type for forwarding and recording traffic.

   You have to set up a proxy for each type of traffic you want to record. Forwarding and recording traffic with the following protocols is supported:

   - **FTP** - File Transfer Protocol
   - **HTTP** - Hypertext Transfer Protocol
   - **LDAP** - Lightweight Directory Access Protocol
   - **POP3** - Post Office Protocol
   - **SMTP** - Simple Mail Transport Protocol
   - **SOCKS -** When capturing SOCKS traffic, the Silk Performer Recorder automatically detects other protocols like IIOP, FTP, POP3 and SMTP for recording.
   - **TCP/IP** Custom protocol based on TCP/IP.
6. In the **Listen port** field, enter the port number to which traffic will be sent by the traffic-generating client computer.

   Although any port not in use can be chosen, avoid standard port numbers (usually smaller than 1500).
7. In the **Remote host** area, choose the **Type** of host to which you want the Silk Performer Recorder to forward intercepted traffic.

   Choose one of the following:

   - Select the **Any** option to forward intercepted traffic to the host specified by the traffic-generating client computer.
   - Select the **Proxy/Firewall** option to forward intercepted traffic to a specified proxy or firewall.

     In this case, the host name and IP address specified by the traffic-generating client computer are ignored.
   - Select the **Specified** option to forward intercepted traffic to a specified remote host.

     In this case, the host name and IP address specified by the traffic-generating client computer are ignored.

8. If you selected the **Specified** option for a remote host, check the **Secure** check box to record and forward Secure Socket Layer protocol traffic.

   If this option is selected, HTTPS traffic will be recorded instead of HTTP traffic, for example.
9. In the **Host name or IP address** field, enter the host name or IP address of a specific server to which you want the Silk Performer Recorder to forward intercepted traffic.
10. In the **Port** field, enter the port number that a specific server—the one to which the Silk Performer Recorder forwards intercepted traffic—listens to.
11. From the **Certificate** list box, select a client certificate that the Silk Performer Recorder is to present to the Web server on behalf of the real client's certificate.
12. When setting up a SOCKS proxy, use the **Record always at TCP/IP protocol level** area to specify any number of ports for which to disable automatic protocol detection.

    You can enter multiple port numbers and ranges, separated by commas (,); a range specification consists of the lower and upper boundary, separated by a hyphen (-).

    Whenever a client application sends SOCKS traffic to one of the specified ports, the Silk Performer Recorder records traffic at the TCP/IP level without attempting to detect a familiar protocol first.
13. In the **Suppress recording (only forward data)** area, specify a port range for which you want to suppress the recording of data by the Silk Performer Recorder.

    You can enter multiple port numbers and ranges, separated by commas (,); a range specification consists of the lower and upper boundary, separated by a hyphen (-).

    The data will still be forwarded, but no longer recorded.
14. Click **OK**.
15. To activate a proxy, check the check box to the left of the proxy name, in the **Record** column.
16. Click **OK** to save your settings.

Configuring Services

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Click the **Services** tab.
4. Use the **Integrated Web server** area to change the port number of the Web server that is integrated into the Silk Performer Recorder.

   The default address of the server is `http://localhost:19100` (where `19100` is the default port number). You must have the Silk Performer Recorder running to use this Web server.
5. In the **Port** field, enter the port that the integrated Web server listens to.

   The default is `19100`. Using the integrated Web server, you can retrieve the root CA certificate that signed the Silk Performer Recorder server certificate. The root CA certificate may be downloaded from this Web server and installed into the Web browser to avoid security warnings by the browser.
6. Select the **Autostart recording** option in the **Recording** area to start recording traffic and function calls automatically when the Silk Performer Recorder is started.

   If this option is disabled, you must use the **Start Recording** button on the Silk Performer Recorder toolbar to begin recording.
7. Click **OK** to save your settings.

Adding a Recording Rule

By default, there are no predefined recording rules. You must script new recording rules manually or create them via the UI, as explained below.

1. Navigate to **System Settings** > **Recorder** > **Recording Rules**.
2. Click **Add**. The **Recording Rule Properties** dialog opens.

3. From the **Template** list box, select a rule template. The **Description** text box explains what the selected template does.
4. Edit the default **Rule Name** and **Description** as required.
5. Check the **Active** check box to activate the rule.
6. Edit the property values as required.
7. Click **OK**. The new rule appears on the **Recording Rules** page.

Editing a Recording Rule

1. Navigate to **System Settings** > **Recorder** > **Recording Rules**.
2. Select an existing rule in the **Rule Name** text box.
3. Click **Edit**.The **Recording Rule Properties** dialog opens.
4. Edit the **Rule Name** and **Description** as required.
5. Check/uncheck the **Active** check box to activate/deactivate the rule.
6. Edit the property values as required.

   Select a property row to view a description of the property in the description text box at the bottom of the dialog.

   Note: Grayed-out properties are read-only.
7. Click **OK** to save your changes.

Agents Settings

To access the Agents settings, click **Settings** > **System** > **Agents**. The following tabs are available:

- **Agent Pool**: Here you can manage your pool of agents. You can add and remove LAN and WAN agents, set a number of agent properties, and export the agent pool.
- **Advanced**: Here you can instruct Silk Performer to use a particular user account for your agents and to create multiple sessions on an agent computer.

Agent Pool

Configuring Agent Pool Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Agents** button.
3. Click **Check Availability** to send a remote call to the selected agent and verify that it is available and active.
4. Select an agent computer and click **Manage** to set the agent's configuration options, including the configuration of multiple IP addresses. The **System Configuration Manager** opens.
5. Click **OK** to save your settings. Repeat the previous step for each agent computer that you want to configure.
6. Back on the **System Settings** dialog box, click **OK** to save your agent pool settings.

Adding a Computer to the Agent Pool

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Agents** button.

   Tip: Alternatively, click **Assign Agents** on the Silk Performer workflow bar. Then click the **Configure Agent Pool** link on the **Setup Agents** dialog box.

3. On a LAN network:
   a) Click the **Add LAN Agent** button. The **Add LAN Agent** dialog appears, listing all compatible agent computers that are available on the subnet.
   b) From the **Running agents list** of computers in your local network, select the computer(s) you want to add to your agent pool.
   c) Click the **Add** button.
4. On a WAN network:
   a) Click the **Add WAN Agent** button. The **Add WAN Agent** dialog appears.
   b) Enter connection parameters for the WAN agent.
   c) In the **Host name or IP address** field, enter the computer's host name or IP address.

      Tip: If you want to add a computer from Windows domains outside your local network, click **[...]**.

      The **Browse for Computer** dialog appears.
   d) Select the computer you want to add to the agent pool and click **OK.**
   e) Click **OK** again on the **Add WAN Agent** dialog.

Agent Connection Properties

Before a Silk Performer agent can be assigned manually to a load test, it has to be added to the agent pool. To open the agent pool, click **Settings** > **System** in the Silk Performer menu and click the **Agents** icon.

You can add agents to the agent pool and remove them. To modify the connection settings, select an agent and click **Properties**.

- Enable **Encryption (SSL)** to force the controller to secure the connection to the remote agent.
- Enable **Authenticate** and specify a **Password** to connect to the remote agent using a password. The password must be identical to the password you specified in the System Configuration Manager.
- When **Encryption (SSL)** is enabled, specify the port the controller uses to connect to the agent in the **Connect to secure port** field. The port must be identical to the agent connection secure port you specified in the System Configuration Manager.
- When **Encryption (SSL)** is disabled, specify the port the controller uses to connect to the agent in the **Connect to port** field. The port must be identical to the agent connection secure port you specified in the System Configuration Manager.
- Enable **Use HTTP proxy server** to connect to the agent using an HTTP firewall. Specify the **Address** and **Port** to which you want to tunnel. These can be obtained from your system administrator. See *Connecting Through HTTP Firewalls*.
- Enable **Use SOCKS proxy server** to connect to the agent using the SOCKS protocol. Specify the **Address** and **Port**. See *Connecting Through SOCKS Firewalls*. If you use both the HTTP firewall and the SOCKS protocol, the agent will first connect to the SOCKS proxy and then connect to the HTTP proxy. See *Connecting Through Multiple Firewalls*.
- Click **Check Connection** to verify that the controller is able to connect to the agent. If you specified a connection through a proxy server, you will be asked for user credentials if the proxy server requires authentication.
- Click **OK** to save your settings.

Removing a Computer from the Agent Pool

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Agents** button.
3. On the **Agent Pool** tab, select the agents that you want to remove from the **Agent Pool** list and click the **Remove** button.
4. Click **OK** on the ensuing dialog box to confirm that you want to remove the agent.

Advanced Agent Settings

By default, remote agents run under the local SYSTEM account. This is because the Silk Launcher Service, which runs under the SYSTEM account, passes on its privileges to the Silk Performer agent and virtual user processes. This concept works fine for protocol web and related technologies. However, for browser-driven web and other technologies that need access to a user registry hive, there are methods to run virtual users under a particular user account.

Using a Particular User Account for a Single Agent

With this option you can configure a specific agent to run its virtual users under a specific user account. If you want to run all your virtual users on all agents under a particular user account, use a different option (see *Using a Particular User Account for All Agents*).

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager**. The **System Configuration Manager** dialog displays.
2. Click the **Applications** tab.
3. Select Silk Performer in the group box on the left side of the dialog box.
4. Click **Change** next to the **Account** text box and type the account name of a user on the agent workstation who has permission to execute remote tests.
5. Click **Change** next to the **Password** text box and type the corresponding password.
6. Click **OK**.

Using a Particular User Account for All Agents

You can specify particular user account credentials in a central location to be used for all agents, except for cloud agents which always use a dedicated user. This feature allows agents to be launched under a specific user account rather than the default SYSTEM account. For technologies such as browser-driven web, it is useful and recommended to run the virtual users in the context of a user account instead of the Windows built-in SYSTEM account. When you enable this setting, make sure that the specified user account is available on all the agent machines that are used for the load test.

By default agents are assigned to the SYSTEM account, but you also have the option to specify a dedicated user account to an agent through the **System Configuration Manager** (see *Using a Particular User Account for a single Agent*). Using this method to assign a specific user account can be cumbersome if you have a large amount of agents, because you have to specify user account credentials separately for each agent.

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

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Agents** icon and the **Advanced** tab.
3. Enable **Use particular user account**.
4. In the fields **Username** and **Password**, specify the credentials for a user with permission to launch the agents to test.
5. Click **OK** to save your settings.

Note: Any settings that are defined in the **System Configuration Manager** for agents will be overridden by this setting on the **Advanced** tab.

Note: If a Remote Desktop Server is installed on the agent, it is recommended to use the default SYSTEM account. If you used a particular user account instead and the session was unexpectedly disconnected, the agent's session including the agent would be ended. This is true if "End a disconnected session" is activated in the Remote Desktop Server configuration.

Distributing Virtual Users Over Multiple Windows Sessions

Windows operating systems do not dedicate all available resources to a single Windows session. For load testing with UI-based technologies such as browser-driven web or SAPGUI, Silk Performer allows you to distribute virtual users over several Windows sessions. This allows you to make better use from resources of powerful hardware. Virtual users that execute Java or .NetFramework code that is not designed to run in multiple instances can be spread over several Windows sessions.

Unlike with GUI-level testing, where each virtual user creates its own Windows session, this setting allows you to run more than one virtual user within a session. How many sessions an agent machine can handle depends on its hardware resources, in particular on the CPU and memory. This setting is available for the following technologies: browser-driven web, Citrix, SAPGUI, Java Framework.

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Agents** icon and the **Advanced** tab.
3. Enable **Create multiple sessions**.
4. Specify the **Maximum sessions** Silk Performer is allowed to create.
5. In the fields **Username** and **Password**, specify the credentials for the Windows sessions that are to be created.
6. Click **OK** to save your settings.

When you start a load test, Silk Performer creates multiple Windows sessions, logs in to the sessions using the credentials, and starts virtual users within the sessions.

Note: When you enable this setting, make sure that your agent machines are set up identically. Also, the agent machines must be equipped with the Remote Desktop Services (RDS) with appropriately set up licensing options. The Windows sessions are opened locally from within the Remote Desktop Services host. Thus, *per device* licenses are better suited, although *per user* licenses work as well.

Tip: The number of virtual users you can run on an agent machine can be read from the agent's capabilities. These default capabilities are calculated based on the CPU and memory. However, a realistic number for your specific environment and conditions can only be determined if the user types (the script and settings) are taken into account. The **Evaluate Agent VUser Capacity** dialog helps you to determine a good maximum number of virtual users for a particular agent and user type. For more information, read *Evaluating Agent VUser Capacity*.

Java Settings

You can configure your Java settings either through the system settings or the profile settings. System settings serve as default settings for new projects and new profiles within existing projects. During a running test, virtual users use the Java settings of the profiles they are assigned to.

Use the general Java settings to specify the JDK version, its location on your hard drive, and the class path. When a class path contains well-known locations such as a project directory or data directory, these path parts are automatically replaced with placeholders. Placeholders are useful when migrating projects or running virtual users on agent machines where the directories have different names than on the controller machine.

Note: When data files have typical Java archive file extensions, such as `.jar` and `.zip`, Silk Performer offers to add the files to the class path of the currently active profile.

Configuring Java Version and Classpath Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Java** icon. The **General** page opens.
3. Specify the directory of the Java home path in the **Java 32-bit home** or **Java 64-bit home** field, depending on what Java architecture you use. You can switch between 32-bit and 64-bit on the **Advanced** tab.

This option enables the Java Virtual Machine to be loaded from a different path than is specified in the `PATH` environment so that you can switch between various Java Virtual Machines without changing the system `PATH` environment.

Note: Silk Performer automatically checks the path you specify here. If the path is not correct, the default Java home path of the operating system is used instead.

The **Classpath** specified for the Java Virtual Machine displays. By default the classpath is set to the system classpath.

4. Click **Check JVM** to verify your Java Virtual Machine configuration settings.

   To ensure that the Java environment is set up properly, the configuration should be tested.
5. Click **New File** to navigate to a class file and add it to the classpath of the Java Virtual Machine.

   Note: Press `Ctrl` or `Shift` to select multiple class files and add them to the classpath.
6. Click **New Directory** to navigate to a directory and add it to the classpath of the Java Virtual Machine.
7. To move selected files up or down in the classpath hierarchy, click **Move Item Up** or **Move Item Down**.
8. To delete a selected file from the classpath, click **Delete**.
9. Click **OK** to save your settings.

Configuring Advanced Java System Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Java** icon. The **General** page opens.
3. Click the **Advanced** tab.
4. In the **Command line options** field, enter any command line options that are to be passed to the Java Virtual Machine.

   Some Java classes require specific Virtual Machine options to run successfully. Tuning parameters may also be specified in VM parameters.
5. Select the **Disable JIT compiler** check box if you want to disable the Just In Time (JIT) compiler of the Java Virtual Machine.

   The JIT compiler can be disabled when required and non-standard VM DLLs can be specified.
6. Check the **Use system classpath** check box to ensure that the system classpath is used.
7. Select **32-bit Java** (default) or **64-bit Java**.
8. If you use **64-bit Java**, specify an **Execution timeout** for the communication between the Silk Performer runtime and the JVM.
9. Click **OK** to save your settings.

The specified Java architecture determines whether your Java test code is running in-process or out of process. This results in the following behavior:

- **32-bit Java** is enabled: When you execute a test, the `jvm.dll` and the required .jar files are loaded dynamically into the `perfrun.exe`. Note that this increases the memory usage of the `perfrun.exe`. Also be aware that during the early phase of a load test, the memory usage might be volatile. This can be misunderstood as a memory leak, but is in fact expected due to the Java garbage collector at work. By default, up to 50 virtual users share the same JVM, which helps reduce memory usage. However, this feature requires that your Java test code is thread-save (especially the static variables).
- **64-bit Java** is enabled: When you execute a test, the Java test code is running in a separate process - it is not loaded into the `perfrun.exe`.

You can also use the BDL function `JavaSetOption` to switch between 32-bit and 64-bit Java. Note that the settings defined in the BDL script override the options defined in the profile settings.

Remote Agent Java Settings

You can configure different Java settings for the controller (the Silk Performer Workbench) and for your remote agents. This means that you do not need to maintain identical Java environments (Java versions, settings, directories) for your controller and agent machines. For agents, a JRE is sufficient, but on the controller, usually a JDK is required.

If a class or .jar file is not found on the agent machine within the specified path, the local data directory will be searched in addition.

There is a corresponding profile settings tab at **Settings** > **Active Profile** > **Java** that pulls its initial values from the remote agent settings at **Settings** > **System** > **Java**. It is the values on the active profile tab that are actually applied to remote agents at test time.

Remote Agents

Silk Performer allows for remote agent computers to be located both in your LAN or somewhere out on the Internet. The communication between the remote agents and the controller can be configured in a distributed secure environment to include secure and encrypted connections.

Communication between remote agents and the controller may be made in a variety of ways with a choice of proxies, including connections made through multiple firewalls and through TCP/IP, HTTP, or SOCKS firewalls. These connections include access through choices for secure or non-secure connections, with or without encryption.

Remote Agent Connection Scenarios

Includes diagrams that illustrate various remote-agent connection types.

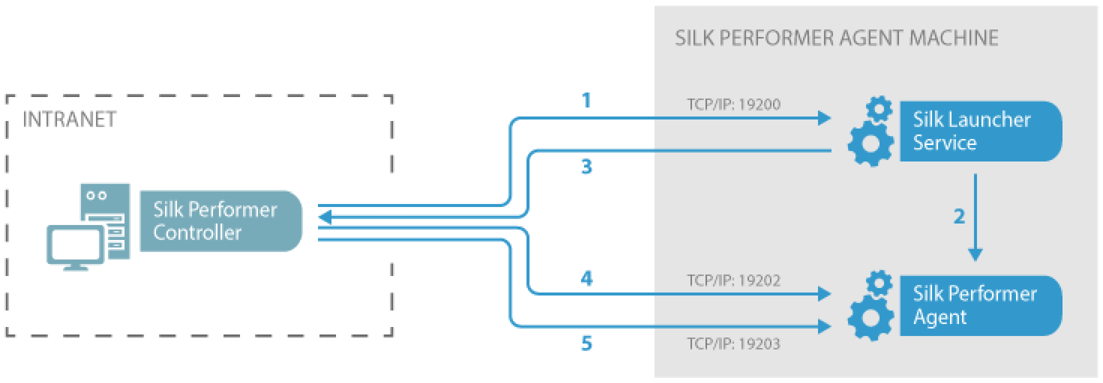
Direct Connection to Remote Agent (default settings)

The following diagram illustrates a direct connection between the controller and the remote agent (no firewalls).

In this environment, the following ports are used to connect to the remote agent:

- `UDP 19200`: Used for the broadcast (hard-coded, does not change)
- `TCP 19200`: Initial unsecured connection to launcher service. Accessible through the **System Configuration Manager** on the agent, accessible through **System** > **Settings** > **Agent Pool** > **Properties** in the Workbench.
- `TCP 19202`: The Silk Performer agent port. This port is the next free port after the maximum port number of the agent connection port (default **19200**) and the secure agent connection port (default `19201`).
- `TCP 19203`: Real-time measure communication port of the Silk Performer agent. This port is only used if real-time measures are enabled. It is the next free port after the agent connection port.

Agent Connection Establishment Without Firewalls



1. Controller opens initial direct connection to Silk Launcher Service.
2. Service launches the Silk Performer agent process (`perfLtcAgent.exe`).
3. Service returns agent's listener port (19202) to the controller.
4. Controller connects directly to the agent on port 19202.
5. Optional: For collecting real-time measure data, the controller connects directly to the agent on port 19203.

Connecting Through TCP/IP Firewalls

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. Click the **Service Status** tab.
3. In the **Port** and **Secure port** fields, enter the numbers of the unsecured port and the secured port.

   By default these settings are `19200` for the unsecured port and `19201` for the secured port, respectively. These port connections must be changed if your firewall blocks them.

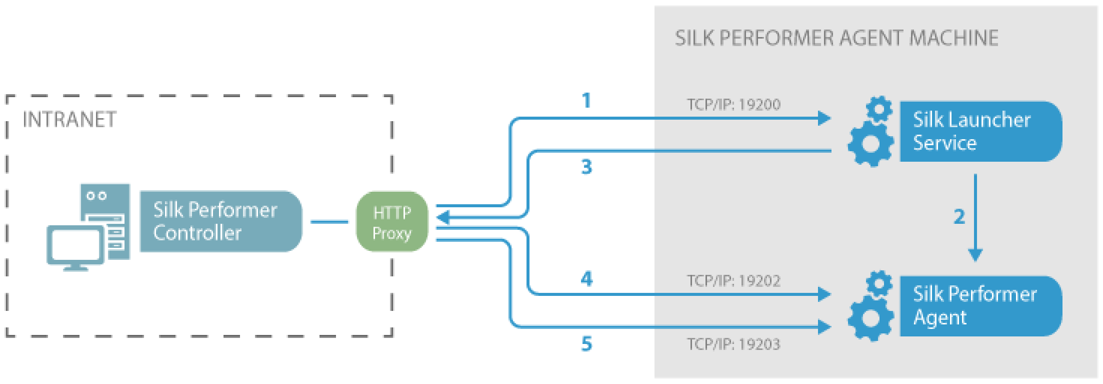
   Ports that you might want to open on your firewall include:

   - 19200: Default connection port to Silk Performer Silk Launcher Service.
   - 19201: Secure connection to Silk Launcher Service (you only have to open this port if you want a secure connection).
   - 19202, 19203, and 19204: These ports are used for communication with Silk Performer Agent, real-time measure service and **System Configuration Manager** Agent. The port assignment follows a first-come, first-served pattern.
4. Click **OK** to save your settings.

Connecting Through HTTP Firewalls

Select the **Use HTTP Proxy Server** setting on the agent **Properties** dialog.

Agent Connection Establishment Through an HTTP Proxy

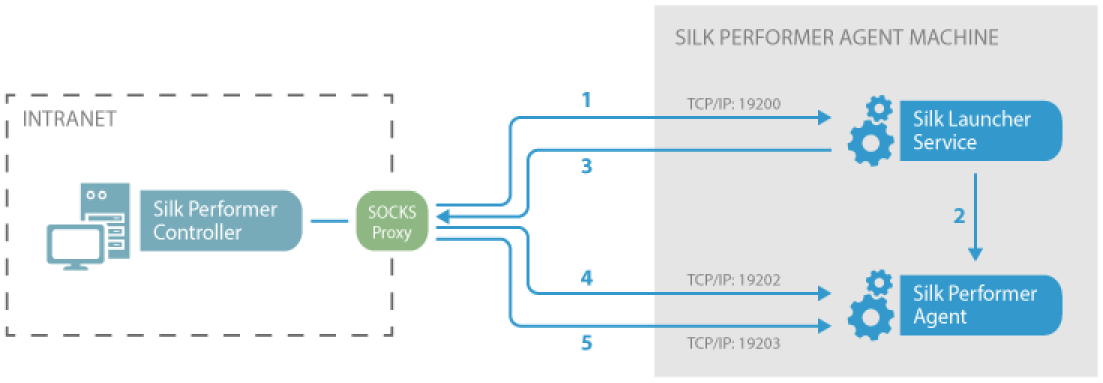


1. Controller opens initial connection to Silk Launcher Service through an HTTP proxy.
2. Service launches the Silk Performer agent process (`perfLtcAgent.exe`).
3. Service returns agent's listener port (19202) to the controller.
4. Controller connects to the agent on port 19202 through the proxy.
5. Optional: For collecting real-time measure data, the controller connects to the agent on port 19203 through the proxy.

Connecting Through SOCKS Firewalls

Select the **Use SOCKS Proxy Server** setting on the agent **Properties** dialog.

Agent Connection Establishment Through a SOCKS Proxy



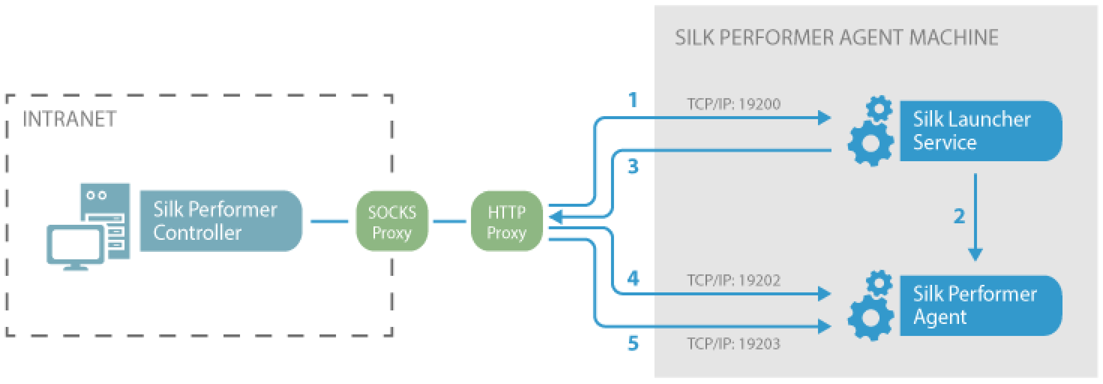
1. Controller opens initial connection to Silk Launcher Service through a SOCKS proxy.
2. Service launches the Silk Performer agent process (`perfLtcAgent.exe`).
3. Service returns agent's listener port (19202) to the controller.
4. Controller connects to the agent on port 19202 through the proxy.
5. Optional: For collecting real-time measure data, the controller connects tot the agent on port 19203 through the proxy.

Connecting Through Multiple Firewalls

You can configure a Silk Performer agent connection so that it uses a SOCKS proxy and an HTTP proxy in a series. In such instances, an agent first connects to a SOCKS proxy, which in turn connects to an HTTP proxy.

Select both the **Use HTTP Proxy Server** and **Use SOCKS Proxy Server** settings on the agent **Properties** dialog.

Agent Connection Establishment Through Multiple Proxies



1. Controller opens initial connection to Silk Launcher Service through a SOCKS proxy then an HTTP proxy.
2. Service launches the Silk Performer agent process (`perfLtcAgent.exe`).
3. Service returns agent's listener port (19202) to the controller.

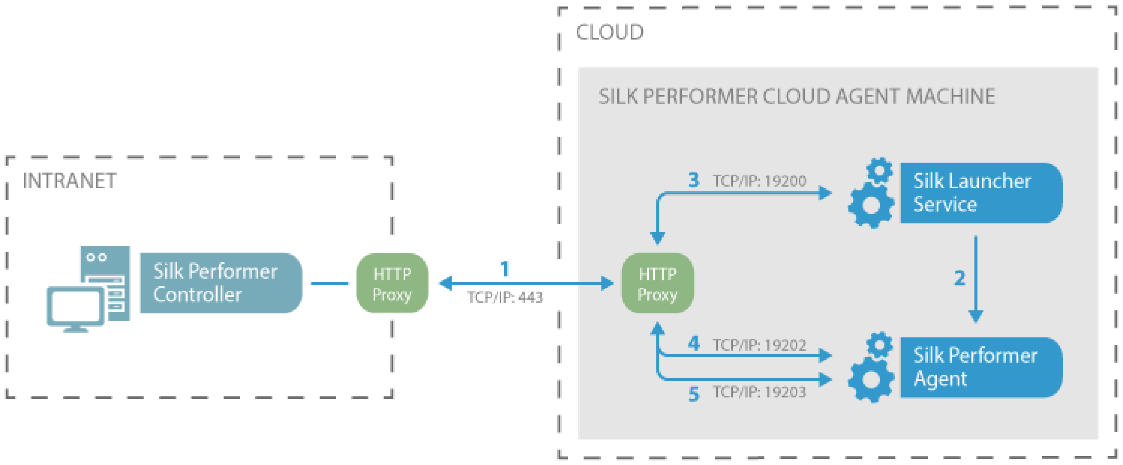
4. Controller connects to the agent on port 19202 through both proxies.
5. Optional: For collecting real-time measure data, the controller connects to the agent on port 19203 through both proxies.

Connecting in the Cloud

Before you begin, contact your sales representative for your Enterprise Cloud Services user name and password.

1. Choose **Tools** > **Cloud Agent Manager**.
2. Type your Enterprise Cloud Services **User name** and **Password**.
3. Click **Connection Settings** if your company requires the use of a proxy.
   a) Select **Enable Proxy**.
   b) Enter the required information in **Address** and **Port.**
   c) Click **OK**.
4. Click **Login**. If your company proxy requires authentication, you may be prompted with another dialog box to log in.

Agent Connection Establishment in the Cloud



1. Controller opens initial connection to Silk Launcher Service through an HTTP Proxy and port 443. Silk Performer automatically uses the default port number of 443 for a secure connection. Connection goes through an additional HTTP proxy if company policy requires it.
2. Service launches the Silk Performer agent process (`perfLtcAgent.exe`).
3. Service returns agent's listener port (19202) to the controller.
4. Controller connects to the agent on port 19202 through the proxy, port 443, and another proxy if required.
5. Optional: For collecting real-time measure data, the controller connects to the agent on port 19203 through the proxy, port 443, and another proxy if required.

Specifying Port Settings for Remote Agents

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. Click the **Service Status** tab.
3. Enter port numbers for the unsecured port and the secured port.

   Port is the TCP/IP listener port for unsecured controller/agent connections (this is the default port used by the controller to connect to agents).

Secure port is the TCP/IP listener port for secure controller/agent connections (a controller can also connect through a secure SSL connection. The controller connects to the secure port if you enable Encryption (SSL) in the **Agent Connection Properties**).

By default these settings are `19200` for the unsecured port and `19201` for the secured port.

4. Click **OK** to save your settings.

Specifying a Password for Remote Agents

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. Click the **Service Status** tab.
3. In the **Password** area, click the **Change** button.
4. Specify a password and then confirm the password.
5. Click **OK**.

   Note: The controller must supply a password to be able to connect to that remote agent.

Configuring Remote Agent Java Settings

1. Navigate to **System** > **Settings** > **Java**.
2. Click the **Remote** tab.
3. Enable **Use different settings for remote agents**. If this option is disabled, the Java settings for the Silk Performer controller will be applied to the agents also. The Java settings for the controller are defined on the **General** tab.
4. Specify the direcotry of the Java home path in the **Java 32-bit home** or **Java 64-bit home** field, depending on what Java architecutre you use. This option enables the Java Virtual Machine to be loaded from a different path than is specified in the `PATH` environment so that you can switch between various Java Virtual Machines without changing the system `PATH` environment. The classpath specified for the Java Virtual Machine is displayed in the **Classpath** field. By default the classpath is set to the system classpath.

   Note: Classpath entries in the **Classpath** field can be edited manually.
5. In the **Command line options** text box, type any command-line options to pass to the Java Virtual Machine.
6. Click **OK**.

Client Certificates

A secure Web server can be configured to refuse clients that do not have a proper client certificate from a trusted certification authority. In such cases, the server asks the client to send its certificate before performing any requests. As long as the client certificate is signed by a trusted CA, the server does not care what the client is. If the Web server requests a client certificate, it is necessary to export the client certificate from the Web browser and place it in the Silk Performer certificate store to be used for recording and replay.

Handling client certificates with the Web browser

A secure Web server can be configured to refuse clients that do not have a proper client certificate from a trusted certification authority. If a Web server requests a client certificate, it is necessary to export the client certificate from the Web browser and place it in the Silk Performer certificate store.

Exporting client certificates

Since the Silk Performer Recorder works as a proxy between a client and a Web server, and the replay engine represents the simulated client itself, successful connection to the server requires a certificate from the client. Therefore the certificate must be exported by the Web browser and imported into Silk Performer.

Note: When obtaining a client certificate from a certification authority, be sure to allow the private keys to be exported with the certificate. Otherwise, the certificate will be useless for the Secure Internet Recorder.

Installing Client Certificates

Client certificates are typically stored on the system or in the web browser's certificate store. To use a client certificate in Silk Performer, export the certificate from your web browser and import it into Silk Performer.

1. From the menu bar, select **Settings** > **Active Profile** .
2. In the shortcut list on the left side, click the **Internet** icon.
3. Switch to the **Security** tab.
4. In the **Client certificate** area, click **Import** and locate a certificate file of the format `.p12` or `.pfx` (for example, `C:\clientcert\johndoe.pfx`).
5. Enter the password you used for encryption when exporting the certificate from your Web browser.
6. Click **OK**.

   This confirmation also creates a certificate file (`.pem`) in the Silk Performer user data directory (for example, `<public user documents>\Silk Performer 20.0\Data\johndoe.pem`).

   Note: Each proxy connection set up in the Silk Performer Recorder can be associated with a client certificate. The Silk Performer Recorder presents the client certificate to the Web server on behalf of the actual client.

Using a Client Certificate for Recording

Before you begin this procedure, make sure that a client certificate is installed in Silk Performer.

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. In the **System Settings - Recorder** dialog, click the **Proxies** tab.
4. Set up a new proxy connection or select an existing proxy of your choice to be associated with a client certificate (for example, SOCKS) and click the **Edit** button.
5. Select an appropriate certificate from the **Certificate** list box in the **Client Certificate** area (for example, `John Doe`).
6. Click **OK.**
7. Restart the Silk Performer Recorder before you begin recording. The Silk Performer replay engine presents the client certificate to the Web server on behalf of the actual client.

Using a Client Certificate for Replay in the GUI

Before you begin this procedure, make sure that a client certificate is installed in Silk Performer.

1. From the menu bar, select **Settings** > **Active Profile** .
2. In the shortcut list on the left side, click the **Internet** icon.
3. Switch to the **Security** tab.
4. In the **Client certificate** area, select an appropriate certificate from the **Certificate** list box (for example, `John Doe`).
5. Click **OK**.

To make the certificate available on remote agents as well, make sure to add the certificate file to the data files of the project. The imported certificates are stored in the Silk Performer user data directory, for example: `<public user documents>\Silk Performer <version>\Data`.

Using a Client Certificate for Replay in a Script

Before you begin this procedure, make sure that a client certificate is installed in Silk Performer.

Use the bdl function `SslSetClientCert()` to enable a virtual user to use the specified client certificate.
For example:

```
SslSetClientCert("johndoe.pem")
```

To make the certificate available on remote agents as well, make sure to add the certificate file to the data files of the project. The imported certificates are stored in the Silk Performer user data directory, for example: `<public user documents>\Silk Performer <version>\Data`.

Exporting a Client Certificate from Internet Explorer

1. From the Internet Explorer menu bar, select **Tools** > **Internet Options** . The **General** page of the **Internet Options** dialog opens.
2. Click the **Content** tab.
3. In the **Certificates** area, click the **Certificates** button. The **Certificates** dialog opens.
4. From the list, select the certificate to export then click **Export**. Internet Explorer automatically starts the **Certificate Manager Export Wizard**.
5. Using the wizard, perform all steps described to export your client certificate.

   Note: You have to export the private key with the certificate for the certificate to be usable with the Silk Performer Recorder.
6. In the **Certificates** dialog, click **Close**.
7. Click **OK** to close the **Internet Options** dialog. The certificate file (`.pfx`) can now be imported by the Silk Performer Recorder.

Exporting a Client Certificate from Firefox

1. From the Firefox menu bar, select **Tools** > **Options** . The **Options** dialog opens.
2. Click the **Advanced** icon.
3. Select the **Encryption** tab.
4. In the **Certificates** area, click **View Certificates**. The **Certificate Manager** dialog opens.
5. From the list, select the certificate to export then click **Export**.
6. In the **Save Certificate To File** dialog, enter a name of the certificate file, select the file type, and click **Save**.
7. In the **Certificate Manager** dialog, click **OK** to close the dialog.
8. Click **OK** to close the **Options** dialog.

Custom Screen Layouts

Different screen layouts are useful for different scenarios. If you are a laptop user developing tests, running tests, and analyzing results, you will probably prefer your laptop's native screen resolution, which may be fairly high. In general the higher the resolution, the more panes you can view at one time. For such scenarios it makes sense to use `High Resolution` screen layout, which enables all panes.

When viewing Silk Performer on a low-resolution device or during the course of a presentation over a projector or Web meeting, your screen resolution will likely be more limited. For such scenarios it may be useful to use normal screen layout.

The customizations you make regarding the panes that you enable/disable and the positions in which you place the panes are saved along with both your high resolution and your normal layouts. For example, if you reposition the **Project** pane while working in high resolution layout and then switch to normal layout, the **Project** pane will appear in the same custom position the next time you return to high resolution layout.

Enabling/Disabling High Resolution Screen Layout

1. Choose **View** > **Screen Layout**.
2. Select `High Resolution` to enable additional Silk Performer view panes for day-to-day testing work on high-resolution screens.

   A checkmark appears next to the `High Resolution` menu option when high resolution layout is enabled.

   Note: To disable high resolution layout and return to normal layout, select `High Resolution`.

Silk Performer screen layout will be redrawn based on your selection.

Resetting Default Screen Layouts

1. Choose **View** > **Screen Layout**.
2. Select `Reset` to restore screen panes and the positions of those panes to their default states.

   The `Reset` command is only applied to your current layout, either high resolution or normal layout, not both layouts.

Silk Performer screen layout will be redrawn based on your selection.

Managing Multiple Versions of Silk Performer

When multiple versions of Silk Performer are installed on a single machine it is possible that the wrong version of Silk Performer will launch when you double-click a project or script file from the desktop. All Silk Performer file types have an entry in the registry that determines what program is to be launched when a specific file type is double-clicked.

By default, the most recently installed version of Silk Performer is the version that is used for all Silk Performer file-type operations. If an older version of Silk Performer was installed most recently, that is the version that will be used when you double-click a Silk Performer file.

There are two ways you can change the default version:

- *Reinstall the latest version of Silk Performer* - All registry entries that control version selection will be updated to reflect the latest version. This is a faster approach and is safer than manually modifying the registry.
- *Modify the registry manually* - Navigate to `HKEY_CLASSES_ROOT\Silk Performer Project Archive File\shell\Open\command` and modify the default value for each file type. You can select a specific version for each file type.

  Note: Each Silk Performer file type has a registry key associated with it. You must modify each entry. While this may be time consuming, it enables you to specify that different versions of Silk Performer be used for different file types.

Managing Load Tests

This section explains the tasks that must be completed to prepare for, run, and analyze the results of a load test .

Outlining Projects

Explains how to define name, description, and application-type-under-test parameters for your project.

Outlining Projects

When you create a Silk Performer load test you must 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.

Outlining a 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, choose the type of application that you want to use in your test.

   Note: If you are testing a Web application, choose the **Web business transaction (HTML/HTTP)** option to create simpler scripts while incorporating advanced functionality. Choose the **Web low level (HTTP)** option if you want to put the highest possible load on your application. Web low level scripts are more complex than Web business transaction scripts, which require more effort to customize. It is recommended that you use the **Web business transaction (HTML/HTTP)** instead of the **Web low level (HTTP)** scripts for browser based applications.
5. Click **Next**.

   Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.

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

Available Application Types

The following application types can be tested with Silk Performer, both secure and unsecured:

Web Browser

Web Browser application types are used for load testing applications that run within a Web browser. Due to the multitude of technologies that browser-based applications are based on, different project types are available for selection.

Web business transaction (HTML/HTTP)

Select **Web business transaction (HTML/HTTP)** to test Web applications with simple scripts that incorporate advanced functionality. This application type should also be used for browser-based applications.

Web browser-driven (AJAX)

Select **Web browser-driven (AJAX)** to use a Web browser (Internet Explorer) to drive your test. This application type is particularly useful when testing a Web application with built-in AJAX logic and testing on the protocol level (HTTP) has proved to be unsuccessful.

Web (Async)

Select **Web (Async)** to test Web applications that use asynchronous communication patterns such as polling, long-polling, and push. The characteristics of such applications is periodic, event-based, or server-triggered content updates without user interaction.

Web low-level (HTTP)

Select **Web low-level (HTTP)** to test a Web application when you want to put the highest possible load on the application. Web low-level scripts are more complex than Web business transaction scripts, which require more effort to customize.

Mobile devices

Select **Mobile devices** to test mobile Web applications. This application type offers simulation capabilities for a variety of mobile devices, such as iPhone, iPad, Android, Windows Phone, and Blackberry. Other mobile browsers can be simulated using custom profiles.

Flex/AMF3 (Adobe)

Select **Flex/AMF3 (Adobe)** for the testing of both AMF0 and AMF3 based on the Adobe/BlazeDS implementation. With this application type, profile settings related to Flex/AMF3, transform HTTP requests/ responses, and JVM are configured automatically.

Flex/AMF3 (GraniteDS)

Select **Flex/AMF3 (GraniteDS)** for the testing of Flex applications and the AMF3 protocol based on the GraniteDS implementation. With this application type, profile settings related to Flex/AMF3, transform HTTP requests/responses, and JVM are configured automatically.

HTTP Live Streaming (HLS)

Select **HTTP Live Streaming (HLS)** to test web applications that stream video or audio data through the HTTP Live Streaming protocol.

Java Over HTTP

Select **Java over HTTP** to test Web applications that make use of Java Object Serialization to transfer objects between client and sever over the HTTP protocol. This communication, based on the exchange of serialized Java objects, basically uses data in a binary format.

WebDAV (MS Outlook Web Access)

Select **WebDAV** to test clients that rely on WebDAV, for example Microsoft Outlook Web Access. WebDAV is a set of methods based on HTTP that facilitates collaboration between users in editing and managing documents and files stored on Web servers.

Silverlight

Select **Silverlight** to test browser-based applications that are built using Microsoft Silverlight.

Note: The workflow for testing Microsoft Silverlight browser-based applications with Silk Performer is the same as the workflow used for testing Web-based applications.

Oracle ADF

Select **Oracle ADF** to test Web applications that are based on the Oracle Application Development Framework, a Java-based framework for building enterprise applications. This project type uses pre-defined recording settings and recording rules to facilitate script customization.

Internet

Select one of the **Internet** application types to test an application that relies upon Internet-based protocols and communication.

Email (SMTP/POP)

Select **Email (SMTP/POP)** to test applications and email clients that use the Simple Mail Transfer Protocol to send and receive email messages.

Directory Server (LDAP)

Select **Directory Server (LDAP)** to test servers that use the Lightweight Directory Access Protocol (LDAP), for example Active Directory.

Radius

Select **Radius** to test applications or devices that use the RADIUS (Remote Authentication Dial In User Service) networking protocol for authentication, authorization, and accounting to manage network access.

FTP

Select **FTP** to test applications that use the File Transfer Protocol (FTP), the standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet.

TCP/IP based applications

Select **TCP/IP based application** to test applications that use proprietary protocols based on TCP/IP.

Mixed Protocols

Select **Mixed Protocols** to test applications that use multiple protocols, such as a combination of HTTP, ODBC, and other protocols.

Terminal Emulation

Silk Performer support for terminal emulation applications enables the recording of terminal-emulator traffic based on the Telnet protocol ("green screen" applications). Supported terminal types include VT100 and VT200 (UNIX, IBM AS400) and IBM mainframes accessed via TN3270(E) & TN5250. A prerequisite for recording terminal-emulator traffic is an installed terminal-emulator for accessing host applications.

Note: For full details regarding Silk Performer support for terminal emulation applications, see `Miscellaneous Tutorials`.

TN3270

Select **TN3270** to test an IBM mainframe application that is accessed using a TN3270(E) terminal emulator.

TN5250

Select **TN5250** to test an IBM mainframe application that is accessed using a TN5250 terminal emulator.

VT100

Select **VT100** to record an application that is accessed using a VT100 terminal emulator.

VT200+

Select **VT200+** to record an application that is accessed using a VT200+ terminal emulator.

ERP/CRM

Select an **ERP/CRM** application type to test an Enterprise Resource Planning or Customer Relationship Management application.

Enterprise resource planning (ERP) systems integrate management information across an organization. Customer relationship management (CRM) systems organize business processes such as sales, marketing, customer service, and technical support.

SAP

Select one of the SAP ERP and CRM application types to test SAP applications.

SAPGUI

Select **SAPGUI** to test an SAP application that uses a SAPGUI client.

SAP NetWeaver (Web)

Select **SAP NetWeaver (Web)** to test a SAP NetWeaver-based application. For example SAP GUI for HTML or SAP Enterprise Portal applications.

PeopleSoft

Select one of the PeopleSoft application types to test a PeopleSoft ERP/CRM application.

PeopleSoft8

Select **PeopleSoft8** to test a PeopleSoft 8.x application.

Silk Performer offers page-level support for testing of PeopleSoft 8.x applications. This application type adds PeopleSoft-specific enhancements for session handling, detection of application-level errors, easy customization of login data, thinktimes, page timer names, and access to table rows.

PeopleSoft9

Select **PeopleSoft9** to test a PeopleSoft 9.x application.

Silk Performer offers page-level support for testing of PeopleSoft 9.x applications. This application type adds PeopleSoft-specific enhancements for session handling, detection of application-level errors, easy customization of login data, thinktimes, page timer names, and access to table rows.

Clarify 8-10 (Tuxedo)

Select **Clarify 8-10 (Tuxedo)** to test an eFrontOffice application.

Clarify eFrontOffice uses TUXEDO as part of its multi-tier architecture.

Oracle

Select an Oracle Application type to test an application built on Oracle business (non-database) software.

Oracle Applications comprise the applications software or business software of Oracle Corporation. The term refers to the non-database parts of Oracle's software portfolio. Oracle sells many functional modules that use the Oracle RDBMS as a back-end, notably Oracle Financials, Oracle HRMS, Oracle Projects, Oracle CRM, and Oracle Procurement.

Oracle Applications 11i

Select **Oracle Applications 11i** to test Oracle Applications 11i.

Oracle Applications 12i

Select **Oracle Applications 12i** to test Oracle Applications 12i.

Siebel

Select a **Siebel 6** application type or the **Siebel Web Client** application type to test a Siebel-based (Oracle) CRM application.

Siebel 6/DB2

Select **Siebel 6/DB2** to test a Siebel 6-based (Oracle) CRM application that is built on an IBM DB2 RDBMS.

Siebel 6/Oracle

Select **Siebel 6/Oracle** to test a Siebel 6-based (Oracle) CRM application that is built on an Oracle RDBMS.

Siebel 6/SQL Server

Select **Siebel 6/SQL Server** to test a Siebel 6-based (Oracle) CRM application that is built on a SQL Server RDBMS.

Siebel Web Client

Select **Siebel Web Client** to test a Siebel-based (Oracle) CRM application that uses a Web client.

Remedy

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.5

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.5 Patch 04

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.02

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.03

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.04

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.04 SP1

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.04 SP3

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.04 SP4

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.6.04 SP5

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 7.7

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 8.0

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Remedy 8.1

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

RemedyAR 8.1 SP1

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

RemedyAR 8.8

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

RemedyAR 9.0

Select the correct **Remedy** version to test BMC Remedy IT Service Management. BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer*

Web Services

Select one of the **Web Services** application types to test an application which utilizes that Web services software framework. Your environment and prerequisites will determine which of these options is best for your needs. Some Web services only work with specific SOAP stacks, depending on the server-side implementation of the Web service.

XML/SOAP

Select **XML/SOAP** to test Web service applications that utilize the SOAP (Simple Object Access Protocol) specification standards.

.NET Explorer

Select **.NET Explorer** to test Web services, .NET Remoting objects, and other .NET components.

.NET Explorer is a tool that allows you to create test cases through a point and click interface.

Java Explorer

Select **Java Explorer** to test Web Services, Enterprise JavaBeans (EJB), RMI objects, and other GUI-less Java objects.

Java Explorer is a tool that allows you to create test cases through a point and click interface, suited for Java-based technologies.

Database

Select one of the **Database** application types to test an application that relies on common database APIs.

Oracle

Select **Oracle** to test database applications that rely on Oracle APIs. Silk Performer provides Oracle API functions along with Oracle database traffic recording and replaying support for use in testing.

ODBC

Select **ODBC** to test applications that utilize the Open Database Connectivity (ODBC) framework. An application can use ODBC to query data from a DBMS, regardless of the operating system or DBMS it uses.

DB2 CLI

Select **DB2 CLI** to test database applications that use the IBM DB2 CLI interface.

Application Server/Component Models

Select one of the **Application Server/Component Models** types to test an application that relies upon specific middleware.

CORBA (IIOP)

Select **CORBA (IIOP)** to test an application that relies upon CORBA middleware, for example VisiBroker.

EJB (RMI over IIOP)

Select **EJB (RMI over IIOP)** to test an application developed using Enterprise JavaBeans that communicates using RMI over IIOP. This application type enables you to use Silk Performer's record and replay workflow.

EJB (Java Explorer)

Select **EJB (Java Explorer)** to use Java Explorer to test an application developed using Enterprise JavaBeans. This application type enables you to use Java Explorer's visual scripting capabilities to create your test script.

Tuxedo (ATMI)

Select **Tuxedo (ATMI)** to test an application that is built on Tuxedo middleware and the ATMI API.

Tuxedo (JOLT)

Select **Tuxedo (Jolt)** to test an application that is built on Tuxedo middleware and relies on the Jolt interface.

.NET Remoting

Select **.NET Remoting** to test widely-distributed applications that rely on .NET remoting communication. This application type enables you to use .NET Explorer's visual scripting capabilities to create your test script.

Jacada

Select **Jacada** to test an application that relies on Jacada unified desktop and process optimization software.

Oracle

Oracle Forms clients run in Web browsers as applets. By default, a Web browser will download and install a JRE when loading an Oracle Forms application for the first time. For Oracle Forms 10g or earlier, Oracle's JRE for Oracle Forms is named *JInitiator.* Using JInitiator is not mandatory. Once a JRE has been installed, Silk Performer is set up to record Oracle Forms applications.

Oracle Forms 10g

Select **Oracle Forms 10g** to test an application that uses an Oracle Forms 10g client.

Oracle Forms 11g

Select **Oracle Forms 11g** to test an application that uses an Oracle Forms 11g client.

Oracle Forms 12c

Select **Oracle Forms 12c** to test an application that uses an Oracle Forms 12c client.

Oracle Forms 6i

Select **Oracle Forms 6i** to test an application that uses an Oracle Forms 6i client.

Oracle Forms 9i

Select **Oracle Forms 9i** to test an application that uses an Oracle Forms 9i client.

Terminal Services

Select one of the **Terminal Services** application types to test a Citrix-based application which is hosted by Citrix XenApp.

Citrix

Select **Citrix** to test a Citrix application that is accessed through the Citrix Online Plug-In.

Citrix Web Interface

Select **Citrix Web Interface** to test a Citrix application that is launched from a Citrix Web Interface running within Internet Explorer.

Citrix StoreFront/Netscaler Gateway

Select **Citrix StoreFront/Netscaler Gateway** to test a Citrix application that is launched from a Citrix StoreFront server.

.NET

Select one of the **.NET** application types to test Web Services, .NET Remoting objects, NUnit, or another .NET technology.

.NET Explorer

Select **.NET Explorer** to test Web services, .NET Remoting objects, and other .NET components.

.NET Explorer is a tool that allows you to create test cases through a point and click interface.

.NET Framework Using Visual Studio .NET Add-On

Select **.NET Framework using Visual Studio .NET Add-On** to test Web Services or .NET components using the Silk Performer Add-On for Visual Studio .NET.

Java

Select one of the Java application types to test Web Services, RMI, JUnit, or other Java-based technologies.

Java RMI/EJB (recording)

Select **Java RMI/EJB (recording)** to test an application developed using Enterprise JavaBeans. This application type enables you to use Silk Performer's record and replay workflow.

Java Explorer

Select **Java Explorer** to test Web Services, Enterprise JavaBeans (EJB), RMI objects, and other GUI-less Java objects.

Java Explorer is a tool that allows you to create test cases through a point and click interface, suited for Java-based technologies.

Java Framework

Select **Java Framework** to test a Java application using the Silk Performer Java Framework.

The Java Framework enables you to implement user behavior in Java. When testing an existing Java application you do not need to spend much time creating test scripts. The only effort required is embedding existing Java source code into the framework.

Java Message Service (JMS)

Select **Java Message Service (JMS)** to test Java Message oriented middleware applications by replaying the created script.

Frameworks

Select one of the included frameworks to test an application using either the Java Framework, Visual Basic Framework, or .NET Framework Using Visual Studio .NET.

.NET Framework Using Visual Studio .NET Add-On

Select **.NET Framework using Visual Studio .NET Add-On** to test Web Services or .NET components using the Silk Performer Add-On for Visual Studio .NET.

Java

Select **Java** to test a Java application using the Silk Performer Java Framework.

The Java Framework enables you to implement user behavior in Java. When testing an existing Java application you do not need to spend much time creating test scripts. The only effort required is embedding existing Java source code into the framework.

Monitoring

Select a **Monitoring** application type to generate server-side results that can be archived for future viewing and comparison. Monitoring also reveals, locates, and resolves server bottlenecks, allowing you to examine the performance of operating systems and application servers.

Silk Performance Manager - Infrastructure Monitor

Select **Silk Performance Manager - Infrastructure Monitor** to create an infrastructure monitoring project that can be uploaded to Performance Manager for automated execution.

BDL Monitor for Performance Explorer

Select **BDL Monitor for Performance Explorer** to create a BDL monitoring project that can be used for real-time monitoring in Performance Explorer.

Unit Testing

Select a **Unit Testing** application type to perform JUnit, Java, NUnit, or .NET unit testing.

JUnit

Select **JUnit** to perform JUnit unit testing.

The Silk Performer Unit Test Import tool allows you to select specific test methods and automatically generate BDL stub code.

Java Testing

Select **Java Testing** to import a Java class and test its methods using the Java Framework.

NUnit

Select **NUnit** to perform NUnit unit testing.

The Silk Performer Unit Test Import tool allows you to select specific test methods and automatically generate BDL stub code.

.NET Testing

Select **.NET Testing** to import a .NET assembly and test its methods using the .NET Framework.

GUI-Level Testing

GUI-Level application types are used for the load testing of client applications that cannot be tested on the protocol or API levels. Silk Performer controls the actual client's user interface (fat client) through Silk Test (**Silk Test Classic**, **Silk4J**, or **Silk4NET**). During a load test, each virtual user operates in its own Windows session, using Terminal Services.

Silk Test

Select **Silk Test** to perform GUI-level testing against client applications that cannot be tested on the protocol or API levels. Silk Performer controls the actual client's user interface (fat client) through Silk Test (Silk Test Classic, Silk4J, and Silk4NET). During a load test, each virtual user operates in its own Windows session, using Terminal Services.

Legacy

The legacy group contains project types offered in earlier versions of Silk Performer. These project types have been declared obsolete or replaced by other project types, which offer new or better capabilities. To ensure backward compatibility, some of these legacy project types are still available.

Legacy Web business transaction (HTML/HTTP)

Select **Legacy Web business transaction (HTML/HTTP)** to test web applications without capturing the traffic during recording. A script is generated instantaneously, which means that it is not possible to regenerate the script with different settings later on.

Samples

Select one of these preconfigured test script **Samples** to save time in the manual creation of your test script.

Sample script reuse is a timesaving option for generating Silk Performer test scripts manually (and bypassing the standard method, which uses the Silk Performer Recorder).

Sample project files include scripts, include files, data files, profile files, workload definitions, and more.

Internet

Select an **Internet** sample if you plan to manually write a script to test cookies, POP3/SMTP, HTML forms, and more.

Recording Rule with Custom Conversion DLL

Select **Recording Rule with Custom Conversion DLL** to generate a Web project that shows you how to use custom conversion DLLs in recording rules. This is useful when parsed output data requires transformation in order to serve as input for a subsequent operation.

Cookies

Select **Cookies** to set up a sample project that demonstrates cookie support.

Before you can replay the sample `WebCookie01.bdf`, please fulfill the following prerequisites:

1. Access to a Web server with Active Server Pages.
2. Copy the file `cookie_counter.asp` to the Web root of your Web server.

Email

Select **Email** to set up a testing project that simulates POP3 and SMTP email clients.

Form-based File Upload

Select **Form-based File Upload** to create a testing project for HTML-based file upload functionality.

FTP

Select **FTP** to set up a generic preconfigured FTP testing project.

LDAP

Select **LDAP** to set up a generic preconfigured LDAP testing project.

MAPI

Select **MAPI** to setup a testing project that simulates a basic MAPI session.

Java Framework

Select a sample **Java Framework** application type to set up a JDBC, RMI, or RMI/IIOP test project that uses the Silk Performer Java Framework.

JDBC

Select **JDBC** to set up a sample project that implements a simple JDBC client using the Silk Performer Java Framework.

The sample uses the JDBC THIN driver to execute a query that is specified in the associated BDL script. Before you run the sample, configure the Java runtime and compiler profile settings and add the jarfile of the JDBC THIN driver to the classpath.

RMI/IIOP

Select **RMI/IIOP** to set up a sample project that invokes methods of a virtual user class using the Silk Performer Java Framework.

The sample invokes methods of a virtual user class implemented in `ProductManagerServicesUser.java`.

RMI

Select **RMI** to set up a sample project that invokes methods of a virtual user class implemented in `ServiceHelloUser.java`. The RMI service is defined in `ServiceHello.java` and is implemented in `ServiceHelloImpl.java`.

.NET

Select a **.NET** application type to set up a simple .NET Remoting or Web services testing project that uses the Silk Performer .NET Framework.

.NET Remoting

Select **.NET Remoting** to setup a sample project that shows how to test .NET Remoting with the Silk Performer .NET Framework.

The sample tests remote objects that are hosted by a sample remote server (`RemotingServApp.exe`).

Web Services

Select **Web Services** to set up a sample project that shows how to test SOAP Web Services with the Silk Performer .NET Framework.

The sample tests an ASP.NET Web Service on a demo server.

JMS

Select **JMS** to set up a project that shows you how to make use of Silk Performer JMS API functions for configuring, sending, and receiving messages.

ShopIt v6.0

Select **ShopIt v6.0** to set up a project that demonstrates usage of the Silk Performer sample Web e-commerce site.

ShopIt V 6.0 simulates a simple Web e-commerce site with a catalog of camping merchandise that is available for simulated online purchase. Use this application to experiment with Silk Performer Web-application capabilities. ShopIt V 6.0 is designed to generate errors, including missing Web links (due to merchandise being out of stock) and session errors.

XML

Select **XML** to set up a project that shows you how to make use of Silk Performer XML API functions for reading, writing, and modifying XML documents.

Project Profile Settings

When a project is set up, profiles can be added, copied, renamed, or deleted. A project can contain as many different profiles as necessary, each with its own project-specific settings.

In each profile, options can be set to manage how the Silk Performer Recorder generates scripts from recorded traffic. Options can also be set for the protocols that are used during recording, and simulation settings can be defined for script replay. Options for the results files that are generated can also be defined.

Options are available for the following types of network traffic:

- Citrix
- CORBA/IIOP
- Database
- Internet
- Java
- Jolt
- .NET
- Oracle Forms
- TUXEDO
- Web

Specified settings are associated with specific load test projects. It is easy to switch between different projects, edit projects, and save projects so that they can be modified and reused later.

Profile Administration

Manage your project-setting profiles and apply them to your projects.

Adding a Profile

1. Select **Project** > **New Profile** . The **New Profile** dialog opens.
2. Enter a **Name** for the new profile and click **OK**. The **Profiles** folder expands in the **Project** menu tree and the new profile is available.

Activating a Profile

In the **Project** menu tree, right-click the profile that you want to set as the active profile and choose **Set as Active Profile**.

The active profile is displayed in bold text.

Editing a Profile

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. Edit the settings as required.

   For help on editing settings, see help topics related to the application type under test.
4. Click **OK** to exit the dialog and save your changes.

Renaming a Profile

1. In the **Project** menu tree, right-click the profile that you want to rename.
2. Choose **Rename Profile**. The **Rename Profile** dialog opens
3. Type a new name for the profile.
4. Click **OK**.

Deleting a Profile

1. In the **Project** menu tree, right-click the profile you want to delete.
2. Select **Delete Profile**.
3. Click **Yes** on the confirmation dialog to delete the profile.

Copying a Profile

1. In the **Project** menu tree, right-click the profile you want to copy.
2. Select **Copy Profile** from the context menu. A new profile appears in the **Project** menu tree, with the name `Copy of <profile name>`.

Recorder Settings

Configure general script-recording settings and protocol-specific settings for your project.

Setting General Recording Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list on the left, click the **Script** icon.
5. Click the **General** tab and check the **Comments** check box. The Silk Performer Recorder automatically inserts comments into the generated script. Comments are left-delimited statements that use `//` symbols.
6. Check the **Commented functions** check box to mark the function calls that are performed to retrieve return parameters as comments in the recorded script.
7. Check the **Include think time** check box to generate `ThinkTime` function calls in the recorded script.

   The `ThinkTime` function calls produce delays that simulate typing time and thinking time in the recorded script to create a more realistic simulation.
8. In the **Min. think time recorded** text box, type a time in milliseconds to define a lower limit for delays.

   This limit is included as `ThinkTime` function calls in the recorded script. Delays less than the time specified in the **Min. think time recorded** text box are not expanded to `ThinkTime` function calls.
9. In the **Max. line length** text box, specify the maximum number of characters in a script line.

   Use this setting to split large amounts of recorded data into several script lines.
10. *Optional:* In the **Script namespace** text box, type a prefix for the names that are declared in the generated script.

    The user groups, transactions, and variables defined in the script begin with the specified string.
11. Check the **Record passwords encrypted** check box to make the recorder automatically encrypt recorded passwords before inserting them into generated scripts.
12. Click **OK** to save your settings.

Setting Protocol-Specific Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. Click the **Script** icon.
5. Click the **Protocols** tab.
6. In the **Web comments** area of the page, specify details about automatically-inserted comments (for Web functions) in the generated script. The settings for these options are only taken into account when the **Comments** option (**General** tab) for script generation is enabled.

Option	Description
Redirection / Authentication	Have comments inserted into the generated script when a redirection (HTTP 302, HTTP 407) from one URL to another is recorded.
HTTP errors	Have comments inserted into the generated script when HTTP errors (HTTP 4xx, HTTP 5xx) are recorded.
Link, form, custom URL search details	Have comments inserted into the generated script when links, forms, or customs URLs are recorded.
Detailed info for form fields	Have comments inserted into the generated script when hidden, changed, or filled out form fields are recorded. **Note:** If this option is not selected, comments for form fields are still recorded, but the additional detailed information (for example, if value is changed or unchanged) is not.
Custom URL parsing details	Have comments inserted into the generated script when parsing details of custom URLs are recorded.

7. In the **IIOP** area of the page, check the **Generate IIOP Get functions** check box to include functions in your test script that will retrieve the return parameters of CORBA operation calls. You can mark the corresponding function calls as comments.

   Use the **Database** area of the page to specify how SQL statements are displayed and whether to include fetched data in the generated test script.
8. Check the **SQL comment** check box to include SQL statements marked as comments in the generated test script immediately before the parse operation.
9. Check the **Trim SQL** check box to display SQL statements in the generated test script in a more readable format, with line breaks and indentation for SQL keywords.
10. Check the **Fetched data** check box to include the fetched data, marked as comments, in the generated test script immediately after the fetch operation.
11. In the **Max. number of rows** text box, enter the maximum number of data rows that are to be inserted as comments immediately after fetch operations.
12. In the **Max. column width** text box, enter the maximum width of columns inserted as commentary.
13. Click **OK** to save your settings.

Replay Settings

Configure script-replay settings for your project such as workload simulation and error handling.

Configuring Simulation Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the **Simulation** tab, check the **Stress test** check box to disable the use of wait periods that are specified in low-level web functions or invoked by calling think-time functions.
4. Check the **Random thinking time** check box to replace recorded actual thinking periods that are specified in low-level web functions or invoked by calling think-time functions, with random values that follow an exponential distribution.
   - **Exponential distribution** - Click this option to replace recorded think-time periods that are specified in Web functions, or invoked by calling the `ThinkTime` function, with random values that follow an exponential distribution. Due to the nature of exponential distribution, extreme random think-time periods can result. Refer to the `SetRandomThinkTime` BDL function for more details.
   - **Uniform distribution of +/- [] %** - Click this option to replace recorded think-time periods that are specified in Web functions, or invoked by calling the `ThinkTime` function, with random values that follow a uniform distribution within a specified range. Use uniform distribution when extreme think times are undesirable. Refer to the `SetRandomThinkTime` BDL function for more details.
5. Check the **Think time limited to** check box to define a limit for think times. When this option is selected, think times are limited to the specified maximum value. Enter an upper limit (in seconds).
6. Check the **Smooth transaction arrival rate (Queuing Workload)** check box to have each virtual user perform its transactions within the specified simulation interval. With this option selected, the simulation time period and the total number of transactions to be called determine the mean arrival interval of the transactions. When this option is disabled, Silk Performer calculates the delay of each transaction arrival randomly.

   Note: This setting does not affect projects that are used for application monitoring.
7. Check the **Choose transactions randomly** check box to force the simulation engine to execute all transactions defined in load-test scripts in random order, based on the queuing workload model. When this option is disabled, simulated virtual users perform their transactions exactly in the order in which they are defined in load-test scripts.
8. Check the **Stop virtual users after simulation time (Queuing Workload)** check box to cancel users when simulation time expires.

   Note: This setting does not affect projects that are used for application monitoring.
9. Check the **Call end transactions for stopped virtual users** check box to have stopped virtual users execute end transactions before stopping. When this option is disabled, virtual users complete their executions without calling end transactions.
10. Check the **Complete current transactions of stopped virtual users** check box to allow stopped virtual users to complete active transactions before stopping. When this option is selected, virtual users check to see if they should stop at the beginning of each new transaction. Thus active transactions are completed before users stop. When this option is disabled, virtual users check during think and wait times, while in the queue, and at the beginning of each transaction to see if they should stop.
11. Check the **Stop virtual users when errors occur in begin transactions** check box to immediately stop virtual users when errors occur during test initialization (which usually occurs in begin transactions). This setting overrides the **Call end transactions for stopped virtual users** and **Complete current transactions of stopped virtual users** options. This setting is useful for tests that involve connect operations that must be successful before tests can proceed.
12. Click **OK** to save your settings.

Configuring Error-Handling Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.

2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. Click the **Errors** tab.
4. Use the **Severity definition** area to modify the severity of the simulation errors that Silk Performer reports in its **Monitor** window.
5. Click **Add** to add a custom severity definition to the list.

   The **Edit Replay Errors and Warnings** dialog box opens.
6. From the **Module** list box, select the module that might cause the simulation error for which you want to define a custom level of severity.
7. From the **Severity** list box, select one of the following default severities of the error for which you want to define a custom level of severity.

Severity	Description
Success	Error is ignored
Informational	Error is ignored but reported
Warning	Error is ignored but generates a warning
Error	Error is treated as an error

   All errors of the selected severity are displayed in the list box below.
8. In the **Error code** text box, type the code of the error for which you want to define a custom level of severity.

   Alternatively, you can select the respective error from the list located above the text box.
9. In the **Description** text box, type a description for native errors.

   Descriptions of other simulation errors are predefined and cannot be modified by the user.
10. From the **Custom severity** list box, select one of the following custom severities for the specified error.

Severity	Description
Success	Error is ignored.
Informational	Error is ignored, but reported.
Warning	Error is ignored, but causes a warning.
Error	Error is reported, but does not affect the simulation.
Transaction exit	Current transaction is aborted.
Process exit	User simulation is terminated.

11. Click **OK**.
12. To edit the custom severity of an error, select it from the **Severity definition** list box and click **Edit**. The **Edit Replay Errors and Warnings** dialog box opens.
13. Edit the custom severity of the error as explained earlier in this task.
14. To remove a custom severity definition from the list, select it from the **Severity definition** list box and click **Remove**.
15. Click **OK** to save your changes and close the dialog box.

Result-File Settings

Configure result-file settings for your project, including TrueLog, time-series data, client-side monitoring options, and logging.

Setting General Result Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Results** icon.
4. Click the **General** tab.

   Use the **Results files** area to specify the types of result files you want to generate. While a test runs, a separate file is created for each virtual user in the current agent. Since generating results files alters the time measures of tests, these files should only be created for debugging purposes.

   - Check the **Virtual user log files (.log)** check box to generate log files (.log), one for each virtual user in the current agent. These files contain all the function calls that are invoked by the transactions of a specific user.
   - Check the **Virtual user report files (.rpt)** check box to generate report files (.rpt), one for each virtual user in the current agent. These files contain simulation results for each user.
   - Check the **Virtual user output files (.wrt)** check box to generate output files (.wrt), one for each virtual user in the current agent. These files contain the output of write statements used in the test script. An output file is generated for a particular user only if that user executes write statements.
   - Check the **Virtual user report on error files (.rpt)** check box to generate report on error files (.rpt) and error files (.err) for each virtual user only when an error (a specific severity level) occurs. These files contain the simulation results for each virtual user.
   - Check the **Time series data (.tsd)** check box to enable generation of throughput information (viewable via Performance Explorer). You can specify the interval at which data is computed. This setting does not affect projects that are used for application monitoring.
   - Check the **Client-side measures in real-time** check box to enable Silk Performer to retrieve real-time client-side measurements for use in Performance Explorer .

     Note: This option applies only to the current profile.
5. In the **Passwords** area, select the **Hide passwords in logs** check box to ensure that characters are hidden when entered into password fields of HTML forms, and are replaced with asterisks (`*`) in recording and replay log files (`*.xlg` and `*.log`).

   Note: This feature is only supported for the **Web business transaction (HTML/HTTP)** project application type when the browser emulation level **Page-level Web API (HTML/HTTP)** is enabled, which is the default for this application type.
6. Check the **Add transaction name and an identifier to measure names** check box to add the transaction name and an identifier as a prefix to measure names. This setting applies to all measure types beside transaction timers. Enabling this setting facilitates the navigation between reports and scripts as reports contain measures in the order that they appear in the script.
7. Click **OK** to save your settings.

Setting Time-Series Data Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **Results** icon.
4. Click the **Time Series** tab.
5. From the **Computation interval** list box, select the interval at which you want data to be calculated.

   This setting is used for all generated time-series data. Keep in mind that the shorter the interval and the longer the test, the more data that will be generated.
6. Check the **Enable all measure groups** check box to generate high-level throughput information for all the measure groups shown in the list below.

   To choose a particular combination of measure groups for which you want to generate high-level throughput information, uncheck the **Enable all measure groups** check box, and select your options in the measure group list.
7. Specify a **Detailed page timers** generation setting. Choices include:
   - **For the whole page**
   - **For documents only**
   - **Disabled**
8. To define an overview report template for generating overview reports with Silk Performance Explorer for this testing project, click **[...]** on the **Overview report template** portion of the **Time Series** page.
   a) From the **Select Template File** dialog box, select the overview report template (`.ovt`) file that you want to have applied to your project.
   b) Click **Open**.

   The file that you specify here will be used as an overview report template for the generation of all overview reports for this testing project. The template file name will also be preselected on the **Overview Report Template workflow** dialog box when new overview reports are created manually.
9. Click **OK** to save your settings.

Setting Monitoring Options

Silk Performer enables you to define monitoring templates for your projects and to configure projects to automatically launch Performance Explorer when load tests begin.

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Results** icon.
4. Click the **Monitoring** tab.
5. In the **Monitoring options** area, check the **Automatically start monitoring** check box to automatically launch Silk Performer's monitoring facility when the test starts.
6. To automatically use the monitoring template that best suits the project, click the **Use default monitoring template** option button.
   For example, if you are creating a Web project, the template specifies the measurements that are useful for Web load tests.
7. To use a custom monitor template, click the **Use custom monitoring template** option button and perform one of the following steps:
   - Type the name of the custom template file (`.pew`) that you want to use to monitor your server. Silk Performer creates a copy of the standard monitor template.
   - Click the folder icon in the name field to select an existing monitoring template.
8. *Optional:* Click **Edit Custom Monitor Template** to add or remove any monitoring performance data.

   When you click this button, Silk Performance Explorer opens. Perform the following steps:

a) Add or remove any monitoring performance data.
b) Save the Silk Performance Explorer workspace to apply your changes to the template.

9. In the **Performance Monitor integration** area, check the **Compute online performance data** check box to compute data for additional performance measurements to be displayed in the Windows Performance Monitor.

   You can use this data to view concurrent users, transaction throughput, sent and received data, and executed SQL statements.
10. Click **OK** to save your settings.

When you perform a test, Silk Performer displays the server performance data that is relevant to the type of server under test.

Setting TrueLog Traffic-Capturing Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Results** icon.
4. Click the **TrueLog** tab.
5. To enable generation of TrueLog files (`.xlg`) during recording, check the **TrueLog files (.record.xlg)** check box.

   TrueLog files contain all traffic that is sent to and received from the server, and the transactions and function calls that are to be generated in the test script.

   Note: TrueLog files should only be created for debugging purposes.
6. Check the **TrueLog files (.xlg)** check box to generate a TrueLog file (`.xlg`) for each virtual user.

   These files contain all the function calls that are invoked by the transactions of a specific virtual user.
7. To enable generation of TrueLog On Error files (`.xlg`), check the **TrueLog On Error files (.xlg)** check box.

   Note: Truelog On Error files can be turned on for even large tests where you expect a moderate number of errors without major impact on replay performance.
8. Specify how much information is to be stored in the TrueLog using the **Store TrueLog** options.

   The options in the **TrueLog On Error** area of the dialog box are only available when TrueLog On Error file generation has been enabled above.

   - Click the **For one transaction** option button to log the entire transaction in which each error occurs. This option requires a significant amount of memory.
   - Click the **Based on content history** option button to log a minimum of relevant data (based on data kept in the history) to the TrueLog. If low-level web calls are used, this setting will have no impact on the size of stored data. This selection requires minimal memory.
9. From the **Generate TrueLog On Error for** list box, select the severity level at which TrueLogs are to be generated (`Errors`, `Severe errors`, or `Warnings`).

   The default severity level is `Errors.`
10. Click the **Log all embedded objects** option button to log all embedded objects (images, java files, etc) in the TrueLog. Logged objects appear in the cached documents section of the TrueLog.
11. Click the **Exclude cached embedded objects** option button to only log embedded objects (images, java files, etc) that have been downloaded within the scope of the current log in the TrueLog. Objects

may not be logged if they appear in the current log as a cache hit. Such objects may have been downloaded outside the scope of the current log. They may also no longer be present in the cache.

12. Click the **Exclude all embedded objects** option button to exclude all embedded objects (images, java files, etc) from being logged in the TrueLog.

    Note: Optimal scalability is achieved by selecting this option.

13. Click **OK** to save your settings.

Setting General Logging Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Results** icon.
4. Click the **Logging** tab.
5. Check the **Log all** check box to include all available types of information in the generated log files.

   These files contain information about timers, transactions performed by the virtual user, and function calls invoked by the transactions. Keep in mind that creating log files can alter time measures.
6. Use the **General** area to specify which types of data should be included in log files.

   - Check the **Transactions** check box to include an overview of the transactions the virtual user performs during the simulation.
   - Check the **Timers** check box to include custom timer information.
   - Check the **API calls** check box to include an overview of the Silk Performer API calls that the user performs during the simulation.
   - Check the **Informational** check box to include additional information, for example, connection information.
   - Check the **Data** check box to include the data that is exchanged with the server.
   - Check the **Nonprintable characters** check box to include nonprintable characters, represented as dots.
7. Click **OK** to save your settings.

Setting Internet-Specific Logging Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Results** icon.
4. Click the **Internet logging** tab.
5. Use the **Data sent to server** area to specify the level of detail at which data that Silk Performer and the Recorder send to the server are written to the log file.

   - Check the **HTTP request header from client** check box to write all client request headers that are sent to the server. This option is only available when recording HTTP traffic (not for replay).
   - Check the **Header** check box to write all the headers that are included in requests sent to the server.

    - Check the **Printable data** check box to write all printable data that is included in requests sent to the server.
    - Check the **Binary data** check box to write all the binary data that is included in requests sent to the server.
    - In the **Max. number of lines** text box, specify the maximum number of lines that can be written to the log file when lines containing request data are sent to the server.
6. Use the **Data received from server** area to specify the level of detail at which the data that Silk Performer and the Recorder receive from the server is written to the log file.
    - Check the **HTTP response header to client** check box to write the client HTTP response headers (received from the server) to the log file. This option is only available when recording HTTP traffic (not for replay).
    - Check the **Header** check box to write all the headers that are included in responses received from the server.
    - Check the **Printable data** check box to write all the printable data included in responses that are received from the server.
    - Check the **Binary data** check box to write all binary data that is included in responses received from the server.
    - In the **Max. number of lines** text box, specify the maximum number of lines that can be written to the log file when lines containing response data are received from the server.
7. Check the **Images, applets, and Java classes** check box to write all the images, applets, and Java classes that are received from the server. When this option is selected, these items are represented as binary data.
8. Check the **TCP/IP connection information** check box to write additional information to the log file; for example, when connections to the server are established or closed.
9. Click **OK** to save your settings.

Setting Hook Logging Options for the Recorder

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

    Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

    The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Results** icon.
5. Click the **Hook logging** tab.
6. Use the **Hooking area** on the Silk Performer Recorder's **Log** page to display all the function calls that the client application performs and the parameters that the client application passes to function calls.
    - Check the **Function calls** check box to display all the function calls that the client application performs.
    - Check the **Function parameters** check box to display all the parameters that the client application passes to function calls. To enable this option, you must first check the **Function calls** check box.
7. Click **OK** to save your changes.

Enabling Real-Time Client-Side Monitoring

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. Click the **Results** group icon.
4. Click the **General** tab.
5. Ensure that the **Client-side measures in real-time** check box is selected.
6. Click the **Client-side measures** tab.

   You may have to use the buttons in the upper right-hand corner of the dialog box to scroll through the available tabs.
7. Set the **Computation interval** to `5 sec`.
8. Check the **Enable all measure groups** check box to enable monitoring of all measure groups.

   Alternatively, you can check specific measure types from the list.
9. Click **OK** to save your settings.

Internet Settings

Configure Internet-related settings for your project, including network simulation, host-server, security, and recording options.

Configuring Network Optimization Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Internet** icon.
4. Click the **Optimization** tab.
5. From the **Optimize network settings for** list box, select the type of simulation for Silk Performer and WinSock buffers.

   This setting determines the size of the buffers that are used to transfer data both to and from the server. Large buffers provide good performance; however, Silk Performer needs more memory to achieve this performance. Small buffers may be useful on driver machines with less memory. Note that, when using modem simulation, small buffers create more realistic simulation.
6. Check the **Client IP address multiplexing** check box to enable IP address multiplexing.

   Note: To assign individual IP addresses to virtual users, you first have to set up multiple IP addresses for the network device. You do this using the Silk Performer System Configuration Manager. Following that, you can assign each of the IP addresses to a different virtual user.
7. Check the **Passive FTP semantics** check box to enable passive FTP semantics, which may be useful when connecting through firewalls or proxies.

   This option only affects FTP data transfers.
8. In the **Connect attempts** text box, enter how often a simulated user is to attempt to connect to the server before an error is reported.
9. In the **Connect timeout** text box, specify for how long a simulated user is to attempt to connect to the server before a timeout is reported.
10. In the **Send timeout** text box, specify for how long the simulated user is to attempt to send data to the server before a timeout is reported.

11. In the **Receive timeout** text box, specify for how long the simulated user is to attempt to receive data from the server before a timeout is reported.
12. Click **OK** to save your settings.

Configuring Network Emulation Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Internet** icon. The **Emulation** pane displays.
4. Select a **Connection Type**: `Mobile`, `Wired`, or `Wireless`, and a specific technology (like `LTE`). Depending on your selection, Silk Performer will provide specific default values for the bandwidth, latency, and packet drop.
5. Enable **Use Network Emulation Driver** only if the network emulation driver is installed on your agents. You can then adjust the **Latency** and **Packet Drop** settings to your needs. If the driver is not installed on your agents, do not enable this option. In such a case, the **Latency** and **Packet Drop** settings will not be used.
6. Select a **Distance To Server**. Depending on your selection, the **Latency** changes.
7. You can manually adjust the values for the settings **Bandwidth Down**, **Bandwidth Up**, **Latency**, and **Packet Drop**.
8. You can click **Reset** to restore the default values for your selected connection type.
9. You can click **Default** to completely restore all values on the **Emulation** pane.
10. Click **OK** to save your settings.

Configuring Host Server Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Internet** icon.
4. Click the **Hosts** tab.
5. In the **Standardhost** area, specify the server you want to use as the standard host.

   You can use either a domain name or an IP address. The name you specify is used whenever the `standardhost` variable is included in test scripts. Optionally, you can use the **Port** text box to specify a port number to use with the standard host.
6. In the **SOCKS4 proxy** area, use the **Proxy name or IP address** and **Port** text boxes to specify the host name or IP address as well as the port number of the SOCKS4 proxy through which all Internet traffic must be directed.

   Use this option when the traffic in a test must negotiate a firewall, for example.
7. Click **OK** to save your settings.

Configuring Security Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.

2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Internet** icon.
4. Click the **Security** tab.

   Use the **Protocol** area to specify protocol-specific settings used for secure connections to remote servers.
5. From the **SSL version** list box, select the Secure Socket Layer protocol version used when establishing a secure connection to the server.
6. From the **Encryption strength** list box, select the encryption strength for data transferred to and from the server over a secure connection.
7. From the **Certificate** list box in the **Client certificate** area, select the client certificate that Silk Performer is to present to the Web server on behalf of the actual client.

   To use a client certificate, you must first export it from the Web browser and place it in Silk Performer's certificate store.
8. To view detailed information about the client certificate currently in use, click **View**.
9. To import a new client certificate from a certificate file (`.p12` or `.pfx`), click **Import**.
   a) Use the **Open Certificate** dialog box to navigate to and select the new client certificate.
   b) Click **OK**.
10. To remove the currently selected client certificate from the list box, click **Remove.**
11. Check the **Force secure connection** check box to use the Secure Socket Layer protocol for script replay.

    This is equivalent, for example, to replacing all the HTTP schemes with HTTPS in the URLs of low-level web functions. It can be used to simplify how tests are used to compare performance with and without the use of secure connections.
12. Click **OK** to save your settings.

Configuring Recording Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Internet** icon.
4. Click the **Recording** tab.
5. In the **SMTP** area, check the **Enable SMTP data recording** check box to record mail body data exchanged between the client application and the mail server.

   To record SMTP traffic, set up an appropriate proxy connection using the **Proxy** page (**Web** settings).
6. Use the **Recorder server certificate** area to specify which server certificate you want the Silk Performer Recorder to present to the client on behalf of the actual server.
7. Check the **Use custom server certificate** check box to have the Recorder present a custom server certificate to the client instead of the default Recorder server certificate.
   a) Click **[...]** to the right of the field to locate the server certificate you want to use.
8. In the **Pass phrase** text box, enter the pass phrase that is to be used, if the server certificate you want the Recorder to use is protected with a pass phrase.

9. Check the **Send root CA during SSL handshake** check box to have the Recorder send the root CA certificate during the SSL handshake.

   The root CA certificate may be requested by a client to authenticate the certificate authority that signed the Recorder server certificate.

   a) Click **[...]** to the right of the field to locate the root CA certificate that you want to use.
10. Click **OK** to save your settings.

Web (Protocol Level) Settings

Configure Web-related settings for your project, including browser, proxy, recording, and verification options.

Configuring Browser Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Web** icon.
4. Click the **Browser** tab.

   Use the **Browser type** area to specify browser-specific settings.
5. From the **Browser** list box, select the Web browser you want to use for your simulation.

   The selection you make determines the format of the header information included in your HTTP requests and the threading model used for simulation.

   Note: For mobile Web application testing (iPhone, iPad, Android, Windows Phone or Blackberry) you can change the user agent string used for recording.
6. Click the **Limit connections per server** option button to limit the connections on a per-server basis as real browsers do (for example, two connections to one server, four connections to two servers).

   Note: It is not recommended that you use this option for low-level Web API, as only the **Max total connections value** is considered.

   - In the **Max concurrent connections to a HTTP 1.0 server** text box, specify the connection limit to every server that uses HTTP 1.0.
   - In the **Max concurrent connections to a HTTP 1.1 server** text box, specify the connection limit to every server that uses HTTP 1.1.
   - In the **Max total connections** text box, specify the absolute maximum number of connections that the emulated browser can open. To perform an accurate connection simulation this value should be above the per-server setting.
7. Click the **Limit connections per virtual user** option button to limit the connections on a per-user basis (for example, four connections to one user, four connections to two users).

   a) In the **Max. concurrent connections** text box, specify the maximum number of simultaneous connections that the client may establish to the server in the case of custom browsers.

      Since the server has to allocate more system resources for each connection, the server's performance decreases as the number of connections increases. As with Web browsers, Silk Performer uses one thread per connection to send and receive data concurrently. For single-threaded (synchronous) operations, set this value to 1. For multi-threaded simulations, choose a value between 2 and 32.

      Note: Values are pre-defined for each selected browser. You can only edit **Max. concurrent connections** if you select **Custom** from the **Browser** list box.

   b) Select the **Single-threaded** option to replay scripts using a single thread.

      This setting forces all low-level web functions to execute synchronously.

8. From the **HTTP version** list box, select the HTTP version of the browser that is to be simulated. This is typically `HTTP/1.0`.
9. Check the **Keep-Alive semantics** check box to set "keep-alive" for HTTP/HTTPS. Keep-alive is a feature that, when supported by both browser and server, allows connections to remain intact after requests and responses are complete.
10. Check the **Content-Encoding** check box to enable Silk Performer to decode compressed data received from the server.

    When this option is enabled, Silk Performer sends an `Accept-Encoding` header to inform the server that encoded data can be submitted (via gzip and compression-deflation methods).
11. In the **User agent** text box, specify user agent information that should be sent to the server when custom browsers are used.

    This is typically the browser name, including platform information. This string is used for the HTTP request header `User-Agent:` that is sent using Silk Performer's low-level web functions.
12. In the **Accept** text box, specify the document types that the server may send to the client custom browsers are used.

    This string is used in the HTTP request header `Accept:` that is sent using Silk Performer's low-level web functions.
13. In the **Additional HTTP headers** text box, specify one or more additional headers, using the exact syntax in the case of custom browsers. Use a new line for each header.

    These headers will be appended to each request that is sent using Silk Performer's low-level web functions.
14. Click **OK** to save your settings.

Proxy Settings

Proxy settings allow you to specify how Silk Performer connect to the Internet. You can define one of the following methods:

- Direct connection to the Internet – Connect to the Internet without using a proxy.
- Connect simulated browsers through a proxy server – Connect to the Internet through a specific proxy server. You can specify separate proxies for HTTP, HTTPS, FTP, and SOCKS traffic.
- Automatic proxy configuration URL – Connect to the Internet through dynamic proxy configuration. Dynamic proxy configuration is a technique used in Web browsers to determine a proxy for each request sent out from the Web browser to the Internet. Depending on the URL or the connection host, a proxy is dynamically determined.

  A proxy auto-config (PAC) file determines the proxy server for each request. A PAC file tells the browser to load its proxy configuration information from a remote JavaScript file rather than from static information you enter directly.

  For more information about the PAC file specification, refer to *http://www.microsoft.com/technet/prodtechnol/ie/reskit/6/part6/c26ie6rk.mspx*.

Configuring Proxy Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **Web** icon.
4. Click the **Proxy** tab.
5. Select one of the following:
   - **Direct connection to the Internet:** Connect to the Internet without using a proxy.
   - **Automatic proxy configuration URL:** Connect to the Internet through dynamic proxy configuration. Specify the location of the PAC file to be loaded using the steps outlined below.
   - **Connect simulated browsers through a proxy server:** Connect to the Internet through a specific proxy server (follow the steps outlined below):

   a) Click the **Advanced** button to open the **Proxy Settings** dialog box.
   b) In the **Internet access** area, configure the **Host name or IP address** and **Port** of the proxy server for each protocol.

      Check the **Use the same proxy server for all protocols** check box to use the same proxy for all Web traffic. Checking this setting enables **Address** and **Port** text boxes on the **Profile - [<profile name>] - Web** dialog box.

      To change the SOCKS proxy settings, click the **Hosts** tab on the **Profile Settings - Internet** dialog box.
   c) In the **Exceptions** area, list any host names, IP addresses, or subdomains for which you do not want to use the proxies specified in the **Internet access** area. This list can contain wildcards (`*`), which cause the application to bypass the proxy server for addresses that fit the specified pattern. To list multiple addresses and host names, separate them with semicolons.
   d) Click **OK** to save your settings.
6. On the **Profile - [<profile name>] - Web** dialog box, click **OK** to save your proxy settings.

Configuring Browser-Simulation Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Web** icon.
4. Click the **Simulation** tab.

   Use the **Simulation** area to set options for realistic simulation of users visiting Web sites.
5. Check the **Simulate user behavior for each transaction** check box to have each virtual users reset their browser emulation after each transaction.

   Depending on the additional option you select, Silk Performer either simulates users who visit a Web site for the first time or users who revisit the site. While users who visit a site for the first time have no persistent cookies stored and no documents cached, users who revisit a page typically have closed their browsers between the Web site visits, have documents cached, and persistent cookies set. Disabling this option lets the virtual user emulate a Web browser that is not closed until the end of the test, thereby reusing cached information throughout multiple transactions.
6. 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.
7. Click the **Revisiting user** option button to generate a realistic simulation of users who revisit a Web site.

   Persistent connections will be closed, and the document history, the non-persistent cookie database, the authentication database, and the SSL context cache will be cleared after each transaction. In such cases, users do not clear the document cache. For more details, review the `WebSetUserBehavior` function in the BDL Function Reference.

Use the **User tolerance** area to adjust the advanced options of the user tolerance simulation.

8. Use the slider to adjust the tolerance level of the simulated user.
9. Click the **Customize** button to open the **Advanced User Tolerance** dialog box where you can alter the individual behavior of the selected user tolerance level.

   Use the **Loading times tolerance** area to specify the user tolerance regarding the loading times.

   a) Check the **If no data arrives after** check box to make the virtual user react if the server does not respond at all in a given time frame.

      Also specify the maximum time in seconds a user is willing to wait if the server does not respond.

   b) Check the **If document is not complete after** check box to make the virtual user react if the first document (root) is received, but the HTML content (frames) is not completely loaded within a given time frame.

      Also specify the maximum time in seconds a user is willing to wait if the first document (root) is received, but the HTML content (frames) is not complete.

   c) Check the **If page is not complete after** check box to make the virtual user react if the HTML content (documents) is received, but the embedded objects are not completely loaded within a given time frame.

      Also specify the maximum time in seconds a user is willing to wait if the HTML content (documents) is received, but the embedded objects are not completely loaded.

   d) Check the **If the pure image load time exceeds** check box to make the virtual user react if the loading time of the embedded objects exceeds a given time frame (The timer starts, when the last HTML document is received).

      Also specify the maximum time in seconds a user is willing to wait for the loading of all embedded objects. This time starts, when the last HTML document is loaded.

   e) Click the **Continue waiting** option button if the user is to continue to wait if the timeout occurs and the page has already been retried the specified number of times; unlimited or until another timeout occurs.

   f) Click the **Abandon** option button if the user should press the stop button (abandon) when a timeout occurs and the page has already been retried the specified number of times.

   g) Click the **Wait additional** option button if the user is to wait an additional period of time if the timeout occurs and the page has already been retried the specified number of times. If this time expires too, the user will press the **Stop** button (ending the load test).

      Also specify the number of seconds that the virtual user is to wait before pressing the stop button.

      Use the **Error tolerance** area to specify user tolerance regarding errors on a page.

   h) Check the **Retry** check box to make the virtual user react to errors in the page.

   i) Specify how often the user is to press the **Refresh** button in the case of errors.

   j) Click the **Page** option button to have the user press the **Refresh** button for any error in the page (for example, a missing image).

   k) Click the **Documents** option button to have the user press the **Refresh** button only when errors in the document portion of the page (for example, a missing frame) occur.

10. Click **OK** to save your settings on the **Advanced User Tolerance** dialog box.
11. Click **OK** to save your settings on the **Profile - [<profile name>]** dialog box.

Setting Browser-Authentication Options

Note: Use browser authentication settings if you want all virtual users to login using the same credentials. If you want each VUser to connect with different login credentials, specify the credentials using project attributes. See Project Attributes Overview in the BDL Function Reference for detailed information or use script customization with data files.

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.

2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Web** icon.
4. Click the **Authentication** tab.
5. In the **Username** text box, enter a user name for authentication when connecting to a protected, remote server.
6. In the **Password** text box, enter a password for authentication when connecting to a protected, remote server.

   Use the **Proxy authentication** area of the dialog to enter a user name and a password for proxy authentication login (required when connecting to the Internet through a proxy server).
7. In the **Login name** text box, enter a user name for proxy authentication.
8. In the **Password** text box, enter a password for proxy authentication.
9. Click **OK** to save your settings.

Setting Browser-Emulation Authentication Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Web** icon.
4. Click the **Emulation** tab.

   Use the **Emulation** area to set options for realistic Web-browser emulation.
5. Check the **Accept cookies** check box to store cookies in Silk Performer's cookie database for each virtual user.

   Note: The cookie database is non-persistent and exists only as long as a load test runs.
6. Check the **Allow automatic redirections** check box to automatically parse redirection responses and redirect requests to the specified server.

   When this option is selected, the user name and password entered on the **Authentication** page are used for automatic authentication to the server whenever necessary.

   a) In the text box to the right of the check box, enter the maximum number of automatic redirections. Silk Performer automatically parses redirection responses and redirects requests to the specified server.
7. Check the **Automatically load images** check box to download images from the server.

   It may be helpful to deselect this option if you are testing dynamic parts of a Web application. Usually images are static objects that stress the network more than the server. Note that ignoring images limits the realism of simulated browser traffic.
8. Check the **Document cache** check box to have Silk Performer emulate a document cache.

   Silk Performer can emulate a document cache by keeping local copies of frequently accessed documents, thus reducing server hits and the time connected to the network. For more details, see the `WebSetDocumentCache` function in the BDL Function Reference.
9. Click the **Once per session** option button to, once per session, check whether or not a page has changed on the Web server since the virtual user last viewed the page.

The term *session* includes the entire sequence of starting a Web browser, surfing the Web, and closing the Web browser. The correlation of a session to Silk Performer transactions can be adjusted using the **Simulate user behavior for each transaction** option on the **Simulation** page.

Once per session works as follows: Each time you start your Web browser, it first looks in its cache before sending a request. The request will either be a *conditional request* (if the document resides in the cache and is requested for the first time in the session), a *normal request* (if the document can not be looked up in the cache), or a *cache hit* (if the document has already been requested in the session). Click the **Every time** option button to always check if a page has changed on the Web server since the virtual user last viewed it.

Documents with a future expiration date are not requested until their dates expire. The **Every time** strategy is slightly different from **Once per session** caching in that it checks for newer documents each time you reload a page. Click the **Never** option button to never check whether a page has changed since the virtual user last viewed it. Selecting this option speeds up the display of pages the user has already viewed.

10. In the **Document history size** text box, enter the maximum number of pages that are to be stored in the history.

    When the maximum number is exceeded, the oldest page is dropped from the history.

    Note: As long as a page is stored in the history, all HTML document related information (links, forms, embedded objects, frames) is kept in memory. As soon as a page is dropped from the history the information is no longer available. Therefore related `WebPageLink` and `WebPageSubmit` calls will not succeed and subsequent page-level calls to the same URL will no longer find the document in the cache.
11. Check the **Emulate DNS lookup** check box to have Silk Performer emulate a DNS lookup mechanism for each virtual user , which can be helpful for DNS-based load balancing.

    When this option is combined with a user behavior option, it allows each virtual user to send its individual DNS query to the DNS server. For more details, review the `WebSetOption` and `WebSetUserBehavior` functions.
12. Click **OK** to save your settings.

Setting Browser-Recording Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Web (Protocol Level)** icon.

   Note: For mobile Web application testing, you can select an alternative user agent string (for example, iPhone, iPad, Android, Windows Phone, or Blackberry) on the **Browser** tab.
5. Click the **Recording** tab.
6. Use the **Browser emulation level** area to specify the level at which you want Web traffic to be recorded for use in your test scripts.

   - Click the **Page-level Web API (HTML / HTTP)** option button to record Web traffic at the browser level. When you choose this option the Silk Performer Recorder generates scripts that call functions provided by the page-level web API.
   - Click the **Low-level Web API (HTTP)** option button to record Web traffic at the browser level. When you choose this option the Recorder generates scripts that call functions provided by the low-level web API.

- Click the **TCP/IP-level API (TCP/IP / HTTP)** option button to record Web traffic on the TCP/IP level. Select this option to capture proprietary protocols carried by HTTP.

7. Select an **Advanced context management level** from the list box.

   You can disable advanced context management, customize the individual settings, or choose one of the presets `Level 1`, `Level 2` (default) or `Level 3`.

   These options are only available when the **Browser emulation level** is set to `Page-level Web API (HTML / HTTP)`.

   - Click the **View Settings** button to view or customize the level of advanced context management.
8. Check the **Automatic page detection** check box to enable the recorder's page-notation technology.

   When you use the page-level Web functions, this feature attaches embedded documents that are loaded by JavaScript code into a Web page. When you use the low-level functions, all the requests that load a web page are wrapped by `WebUrlBeginPage` and `WebUrlEndPage` function calls and so automatically creates page timers and synchronizes the downloading of Web pages for improved accuracy.
9. Check the **Convert URL query strings to forms** check box to convert recorded URL query strings to Web forms and then include the forms in the `dclform` section of the generated test script.

   Disable this option only if the URLs are irregular.
10. Check the **Enable persistent-cookie recording for returning-user simulation** check box to have the recorder generate a `WebCookieSet` function call at the beginning of the current transaction for each persistent cookie that you receive from the Web server.

    When your script is run, each `WebCookieSet` function then emulates a cookie that is persistently stored in the virtual user’s cookie database.

    Note: `WebCookieSet` function calls are marked as comments, so you must remove the comment marks to activate the function calls.
11. Check the **Encapsulate concurrent pages using timers** check box to encapsulate concurrent pages with timers.

    Page timers, normally included in the page-level web functions or in calls of the function `WebUrlBeginPage`, are not scripted. Instead, functions `MeasureStart` and `MeasureStop` are inserted automatically (based on timings). A series of browser activities is considered "concurrent" if there is no browser-idle period longer than the value of the minimum recorded thinktime.
12. Use the **Record additional HTTP headers** area to specify additional HTTP headers you want to record, such as those generated by a custom HTTP client.
13. Click the **Add** button to add the name of an additional HTTP header you want to record, such as one generated by a custom HTTP client. You must know the exact name of the header you want to record to use this feature. The **HTTP Headers** dialog box opens.

    a) Enter the name of the header you want to add to the list, or select one from the list box.
    b) Click **OK**.
14. Click the **Edit** button if you want to edit the name of a selected HTTP header. The **HTTP Headers** dialog appears.

    a) Edit the header name as required.
    b) Click **OK**.
15. Click the **Remove** button if you want to remove a selected additional HTTP header from the list.

    a) Click **Yes** to confirm the deletion.
16. Click **OK** to save your settings.

Setting Verification Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Web** icon.
5. Click the **Verification** tab.

   Use the **Recording** area to specify options for generating verification functions when recording a script based on captured Web traffic.
6. Check the **Record title verification** check box to have the Recorder automatically generate a title verification function (`WebVerifyHtmlTitle`) for each Web page.
7. Check the **Record digest verification** check box to have the Recorder automatically generate a response verification digest function (`WebVerifyDataDigest`).

   Verification functions are of the following types:

   - Click the **All characters** option button to include in the digest all of the characters that are received in HTTP response messages from the Web server.
   - Click the **Printable characters** option button to include in the digest only the printable characters that are received in HTTP response messages from the Web server.
   - Click the **Alphanumeric characters** option button to include in the digest only the alphanumeric characters that are received in HTTP response messages from the Web server.
   - Use the **Verify data of content type** area to list all HTTP response content types for which a response verification digest should be generated.
8. Click the **Add** button to add a new HTTP response content type to be used for response verification. The **Response Content Types** dialog box opens.
   a) Enter the name of the new content type you want to add to the list, or select one from the list box.
   b) Click **OK**.
9. Click the **Edit** button to edit the currently selected HTTP response content type. The **Response Content Types** dialog box opens.
   a) Edit the currently selected HTTP response content type.
   b) Click **OK**.
10. Click the **Remove** button to remove the currently selected HTTP response content type from the list.
    a) Click **Yes** to confirm the deletion.
11. In the shortcut list on the left, click the **Replay** button. The Replay category is displayed.
12. Use the **HTML / XML** area to enable HTML and XML document verification when replaying scripts based on Web traffic captured by the Silk Performer Recorder. Check the **Title verification** check box to enable Silk Performer to verify the title of HTML documents.

    An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the verification fails. The setting used here is valid for both the page-level Web API and the low-level Web API (`WebSetVerificationEx`).

    When this option is selected it triggers the execution of the `WebVerifyHtmlTitle` function in the BDL script.
13. Check the **Table verification** check box to enable Silk Performer to verify that a table within an HTML document contains specified text.

    An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the verification fails. The setting used here is valid for both the page-level Web API and the low-level Web API (`WebSetVerificationEx`).

    This option triggers the execution of the `WebVerifyTable` function in the BDL script.
14. Check the **Digest verification tolerance level** check box to verify the HTML content of a server response.

Use TrueLog Explorer to generate `WebVerifyHtmlDigest` function calls.

Enter the maximum difference in character frequency, quantified in bytes, that may occur between the server response that has been used by TrueLog Explorer to generate the HTML digest and the response that is received during replay. An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the number of differing bytes exceeds this limit.

15. Check the **HTML verification** check box to enable Silk Performer to verify that an HTML document contains specified text.

    An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the verification fails. The setting used here is valid for both the page-level Web API and the low-level Web API.

    When this option is checked, it triggers the execution of the `WebVerifyHTML` and `WebVerifyHTMLBound` functions in the BDL script.
16. Check the **Link checking** check box to check the validity of links in HTML documents received during the test.

    The HTML document is parsed on receipt, and additional requests are sent to check that links are valid. An error message is generated if a target pointed to by a link is invalid. This feature is available only in the page-level Web API.
17. Check the **XML verification** check box to enable Silk Performer to verify that an XML document contains specified values or attributes.

    An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the verification fails. The setting used here is valid for both the page-level Web API and the low-level Web API.

    When this option is checked it triggers the execution of the `WebXmlVerifyNodeValue` and `WebXmlVerifyNodeAttribute` functions in the BDL script.
18. Use the **Data** area to specify options for HTTP response verification when replaying a script based on Web traffic captured by the Recorder. Check the **Data verification** check box to enable Silk Performer to verify that raw data received from the server contains specified data (for example, HTML source code containing specific code).

    An operation (depending on the specified severity: `error`, `warning`, `informational`, or `custom`) will be performed if the verification fails. The setting used here is valid for both the page-level Web API and the low-level Web API.

    When this option is checked it triggers execution of the `WebVerifyData` and `WebVerifyDataBound` functions in the BDL script.
19. Check the **Digest verification tolerance level** check box to have Silk Performer verify server response data during tests.

    Note: The Recorder generates the digests needed for verification.

    In the field to the right, enter the maximum difference in character frequency, quantified in bytes, that may occur between the response message received from the server during recording and the response message received during replay. If the number of bytes that differ exceeds this limit, Silk Performer will report an error. The Recorder creates digests only for the content types that are included in the **Verify data of content type** list.
20. Click **OK** to save your settings.

Setting HTTP-Body Transformation Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **Web** icon.
4. Click the **Transformation** tab.

   Note: You may have to click one of the arrow buttons in the top-right corner of the dialog box to scroll through the available tabs to find the **Transformation** tab.
5. Select a transformation type from the **Type** list box.
6. Optionally, enter additional parameters for the transformation type.
7. Use the check boxes to specify if the transformation should be applied to HTTP request bodies (**Transform HTTP requests**), HTTP response bodies (**Transform HTTP responses**), or both.
8. Check the respective **Log encoded** check boxes to additionally log the encoded HTTP response bodies.
9. Click **OK** to save your settings.

Transformation of Custom Content Types and Encodings

Overview

By default, transformation is enabled for HTTP requests and responses that have the HTTP header `Content-Type` or `Content-Encoding` set to the following values:

- Flex/AMF3 (Adobe, BlazeDS, GraniteDS): `Content-Type` set to `"application/x-amf"`
- Java over HTTP: `Content-Type` set to `"application/octet-stream"` or `"application/x-java-serialized-object"`
- Microsoft Silverlight: `Content-Type` set to `"application/soap+msbin1"` or `"application/msbin1"`
- GZIP POST Data: `Content-Encoding` set to `gzip`

If you need to transform data with a different HTTP Content-Type or Content-Encoding header, this can be customized in the profile settings under **Settings** > **Active Profile** > **Web (Protocol Level)** > **Transformation** in the **Additional Parameters** field.

Enabling Custom Transformation During Replay

1. Select **Settings** > **Active Profile** > **Web (Protocol Level)** > **Transformation**.
2. In the **Additional Parameters** field, type `AdditionalContentTypes=application/<myApp1>;application/<myApp2>`. `myApp1` and `myApp1` are the custom content types of the application under test.

Enabling Custom Transformation During Recording

Note: These steps are not required for `Content-Encoding` customization.

1. Select **Settings** > **Active Profile** > **Web (Protocol Level)** > **Transformation**.
2. In the **Additional Parameters** field, type `AdditionalContentTypes=application/<myApp1>;application/<myApp2>`. `myApp1` and `myApp1` are the custom Content-Types of your application under test.
3. Open the `Documents` folder in your current project directory with a file explorer (`<my documents>\Silk Performer 20.0\Projects\<current project>\Documents`).
4. If this folder contains recording rule files (`xrl`), open them with a text editor.

5. Search the recording rule files for `Content-Type` and duplicate any existing entries with your custom `Content-Type`. Example:

```
<CompareData>
  <ApplyTo>Http.Final.Response.Header.Content-Type</ApplyTo>
  <Data>application/x-amf</Data>
</CompareData>

<Or>
  <CompareData>
    <ApplyTo>Http.Final.Response.Header.Content-Type</ApplyTo>
    <Data>application/x-amf</Data>
  </CompareData>
  <CompareData>
    <ApplyTo>Http.Final.Response.Header.Content-Type</ApplyTo>
    <Data>application/myApp1</Data>
  </CompareData>
  <CompareData>
    <ApplyTo>Http.Final.Response.Header.Content-Type</ApplyTo>
    <Data>application/myApp2</Data>
  </CompareData>
</Or>
```

   Note: If the `<CompareData>` elements are not encapsulated with an `<Or>` element you need to add this.

Setting Terminal Client Options

Note: Silk Performer offers tutorials that walk you through the process of load testing terminal client applications (*IBM Mainframe Applications* and *VT 100+ Applications*). The tutorial is available at **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Documentation** > **Tutorials** > **Miscellaneous Tutorials** .

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Terminal Client** icon.
5. Click the **Telnet** tab.
6. In the **Telnet settings** area, specify one of the following Telnet detection modes:
   - Click the **Always assume Telnet-mode** option button to instruct the recorder to assume that each connection is a Telnet connection. Use this option for troubleshooting purposes only.
   - Click the **Auto-detect Telnet-mode** option button to instruct the recorder to automatically detect Telnet on each new connection (default).
   - Click the **Never assume Telnet-mode** option button to instruct the recorder to assume that no connections are Telnet connections. Use this option for troubleshooting purposes only.
7. In the **Command prompt string** text box, specify a command-prompt string that the recorder uses to script `WebTcpipRecvPacketsUntilData` functions wherever possible.

   This string increases the effectiveness of recorded scripts for dynamic content.
8. In the **Terminal properties** section, select one of the following items from the **Log level** list box:
   - `None`
   - `Error`

- `Normal`
- `Debug`

9. Ensure that the specific terminal type you are testing is displayed in the **Terminal type** list box. This setting was configured automatically when you selected the application type for the project.

   When correctly specified, the recorder scripts the appropriate terminal-type initialization functions and renders the screens to the TrueLog.
10. In the **Configuration string** text box, initialize the terminal emulator with a custom configuration string.

    For available options, refer to your terminal emulator documentation.
11. Specify the following **Screen** values to indicate the appearance of the terminal screen to the recorder:

    - Type a value in the **rows** text box to indicate the height of the terminal screen. An empty value lets Silk Performer detect the height during session initialization.
    - Type a value in the **columns** text box to indicate the width of the terminal screen. An empty value lets Silk Performer detect the width during session initialization.
    - Check the **Colors** check box to specify the default color to use on the terminal screen.

    The supplied values overrule the default values that Silk Performer may detect. If your terminal application uses a custom screen size, make sure to set these values before you record a script in order to replay the script without errors.
12. In the **Host code page** text box, specify the code page that the server uses.

    The code page is used for various data conversions, and all available code pages have been installed with the system. If the required code page is not listed, you can install it. Ensure that the selected code page is also available on each agent computer. To review all installed code pages, or to enable a specific code page, go to **Start** > **Control Panel** > **Regional and Language Options** > **Advanced** . Only the host code pages that are listed and enabled with a checkmark are available.
13. Click the **Advanced** tab and check the **Record basic functions** check box to prevent the recorder from scripting synchronized receive-functions, such as `WebTcpipScreenRecvPacketsUntilCursor` and `WebTelnetScreenRecvRecordsUntilStatus`.

    If this option is enabled, scripted receive functions do not rely on rendered screen content. This option is enabled by default when a terminal type is not specified.
14. Click **OK** to save your settings.

Setting GUI-Level Testing Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **GUI-Level Testing** icon.

   On the **Session** page, the **Session settings** area is used to configure the connection between your console session and remote desktop sessions.
4. Check the **Run Try Script on console** check box to have Silk Performer run a Try Script on the console application.

   When this option is enabled, you can view what Silk Performer does during tests, but you are limited in what you can do on your computer because Silk Test uses your console. If disabled, Try Scripts will connect to and run Silk Test on remote desktop sessions.
5. Check the **Allow usage of console session on agents** check box if you do not have a terminal-server license for all agents, or if your agents run on Windows Home or Windows Professional editions.

   Silk Performer only uses this option if the connection to a terminal services session fails, or if no user credentials are provided.

Note: When this option is selected, you can start only one virtual user on each agent (for agents that have no Terminal Server license and Windows Home/Professional systems).

6. Define a **Connection timeout** (in seconds) to specify how long Silk Performer should wait before a connection is established to remote desktops.

   Set this setting to `0` to have no timeout.
7. Select a **Color Depth** setting.

   The **Color Depth** setting enables you to specify the color depth of terminal server sessions that are used by virtual users to drive applications via Silk Test. You can select from 8 bit (256 colors) up to 32 bit (true color).

   Note: Color depth settings defined using the BDL function `StInitSession` override profile settings configured here. When the optional `uColorDepth` parameter is omitted, profile settings are used.

   The server may limit maximum color depth. The client can not force a color depth setting that is higher than is supported by the server. Please see Terminal Services Configuration documentation for details about your server version's color-depth limitations.
8. Complete the **Username/Password** settings to enable automatic login if you only require a single set of login credentials.

   These credentials will be automatically loaded into the `StInitSession` function. These credentials can be overruled by scripted settings, for example, by providing parameters to the `StInitSession` function using project attributes.
9. On the **Execution** page, the **Execution Settings** area enables you to configure settings for the script and the runtime. Check the **Log Silk Test errors** check box to have errors and warnings that are sent back by Silk Test logged as informational warnings (this option is enabled by default).
10. Select a TrueLog setting for the remote desktop session from the **Generate TrueLog file for** list box:

    - Select `None` (the default) to not have TrueLog written.
    - Select `On Error` to have TrueLog written only when errors are encountered.
    - Select `Custom` when you wish to have TrueLog written based on your custom settings.

    If you select `Custom` TrueLog settings, your TrueLog settings will be read from the Silk Test option file, or they can be set via the `StSetOption` function.

    Note: Even if you select `OnError` or `Custom` from the **TrueLog on SilkTest** list box, Silk Test will only write TrueLog files if you also specify in Silk Performer's settings that TrueLog should be written. To do this, click the **Generate TrueLog Files** button or the **Generate TrueLog On Error Files** button on Silk Performer's toolbar.
11. Select a **Screenshot mode**:

    - **None**: Captures no screens.
    - **Active Window**: Captures the active window of the test application.
    - **Active Application**: Captures the test application and any other window within the test application.
    - **Desktop**: Captures the entire desktop.

    Note: This list box is just enabled for Silk4J and Silk4NET projects. It sets the screenshot mode used for the TrueLog file. This setting applies only to successful playback steps. If playback steps fail, the entire desktop is captured.
12. Specify an **Execution timeout** (in seconds) to specify how long the runtime should wait before test cases execute. Set this setting to `0` to have no timeout.
13. **Results history size** (set to 9 by default) specifies how many test-case results Silk Test can save in a `.res` file.

    Note: By starting a new test, the results in the `.res` file are reset.

14. In the Silk Test Classic**option file** field, browse to and select a Silk Test Classic option file.

Refer to Silk Test Help for full details about option files.

Note: This field is just enabled for Silk Test Classic projects.

15. Click **OK** to save your settings.

Note: When a GUI-level monitor is deployed on a Silk Test Agent with Terminal Services installed, Silk Performer automatically looks up the RDP port used for Terminal Server session creation.

Setting Oracle Forms Options

Oracle Forms-specific settings allow you to modify how Silk Performer interacts with Oracle Forms clients during replay.

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Oracle Forms** icon.
4. On the **General** page, select a **Connection mode** setting that reflects which type of connection the Oracle Forms server accepts.

   Oracle Forms clients can communicate with servers via `Sockets`, `HTTP`, or `HTTPS`.

   Note: This setting can be overridden by the `OraFormsSetConnectMode(connectMode)` BDL function call.

   - Click the **Socket mode** option button when the Oracle Forms server is set up for socket mode.
   - Click the **HTTP mode** option button for Oracle Forms servers that are configured for HTTP connections.
   - Click the **HTTPS mode** option button for Oracle Forms servers that are configured for secure HTTP connections.
5. In the **Connection timeout** text box, specify a timeout period (in seconds) to specify how long emulated clients should attempt to establish communication with the server before they report an exception.
6. To configure additional runtime settings, you can optionally have the virtual user send a heartbeat message to the server at a specified interval. This is useful when the Oracle Forms client is not communicating with the server for a long period of time, for example during a long think time period. Check the **Enable heartbeat with frequency of** check box and enter an interval (in seconds) in the **sec** field.
7. Check the **Automatically wait for application timers** check box to direct the replay engine to wait for application timers to expire after each function call.
8. The **Application timer and window timeout** text box specifies the maximum wait time for expiration of application timers and appearance of windows. Enter a timeout setting (in seconds).

   The replay engine waits for timers to expire after each function call. The function `OraFormsWaitForTimer` can be used to make an application wait for timers to expire. Maximum wait time is specified by this setting. The function `OraFormsWaitForWindow` may be used to have an application wait for windows to appear. This setting specifies the maximum wait time for window appearance.
9. Oracle Forms servers can allow certain client releases. Use the Oracle Forms setting to specify the Oracle Forms client version.

   Oracle Forms clients send the required version when connecting to the Oracle Forms server. Create profiles and change this setting when testing applications deployed on different server versions.

Note: This setting can be overridden by the BDL function call `OraFormsSetInt("INITAL_VERSION", theVersion)`.

10. Click the **Logging** tab.

    Here you can specify that additional properties have their values logged during replay.
11. From the **Log level** list box, select the log level for virtual user logging:

    - `None` - TrueLogs will be generated, though no Oracle Messages will be logged and no detailed information about controls in the log file will be written.
    - `Error` - In addition to the `None` log level, errors that occur during replay are logged.
    - `Normal` - In addition to the `Error` log level, Oracle messages are logged to the TrueLog and the log file.
    - `Debug` - In addition to the `Normal` log level, detailed information about control messages is logged, for example in the **In Body** and **Out Body** tab in TrueLog Explorer. This information is helpful for customizing and debugging your script when comparing it with Try Script runs. This option should also be used when you encounter a problem during replay and you must send your log files to Micro Focus SupportLine for analysis.

    Additional properties to be logged in the TrueLog for each control in your application can also be defined. Such properties are Oracle Forms internal properties that must be defined using their internal names. In most cases you will not need to use this feature, as default properties (name, value) are logged for each control. The `OraForms.bdh` file contains a complete list of all internal properties, most of which are not used by controls and so will generally be ignored if you define them.
12. In the **Additional properties** text box, you can specify other properties for which virtual users are to log values.

    - Click **Add** to open the **Additional Properties** dialog box and add a property that is to have its value logged during replay.
    - Click **Edit** to open the **Additional Properties** dialog box and edit the selected property.
    - Click **Remove** to remove a selected property. Click **Yes** on the deletion confirmation dialog. No further values will be logged for this property.
13. Click the **Measuring** tab.

    Here you can specify a custom measurement level. By default, all available performance metrics are collected during test runs.
14. Check the **Enable all timers and counters for all controls** check box to enable all timers and counters for all actions on controls during replay.

    Alternatively, uncheck the **Enable all timers and counters for all controls** check box and select a specific control from the **Control** list box and select the specific timer/counter types that you want to have applied to that control:

    - `Enable timers` - Measure how long it takes to complete an action on the selected control.
    - `Count round trips` - Count the round trips for each action on the selected control.
    - `Count bytes` - Count the bytes sent and received for each action on the selected control.
    - `Count messages` - Count the messages sent and received for each action on the selected control.

    Note: Alternatively, you can click the **Apply to All Controls** button to apply your timer/counter settings to all controls.
15. Click **OK** to save your settings.

Setting SAPGUI Options

Silk Performer offers a tutorial that walks you through the process of testing SAPGUI applications. The tutorial is available at **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Documentation** > **Tutorials** > **SAP** .

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.

2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **SAPGUI** icon. On the **General** page you can define options for how the SAP Recorder records and replays scripts.
5. Login to SAP systems involves several user interactions. With the **Script logon as a single function** option enabled, the recorder only scripts one function that performs all the required login actions.
6. Check the **Script low level functions** check box to use a basic set of scripting functions instead of object-specific functions.
7. Check the **Script timers** check box to script timers automatically.

   These timers are used to measure SAP API calls.
8. Check the **Attach to existing SAP session** check box to have the recorder attach to an existing session rather than create a new session.
9. Check the **Record window title verification** check box to record `SapGuiSetActiveWindow` functions in such a way that title verifications are automatically performed on activated windows during test runs.
10. In the shortcut list on the left, click the **Replay** button. The Replay category is displayed.
11. In the **Replay timeout** text box, specify a timeout period (in seconds) for automatic shutdown of `SAPFewgsrv` (when not otherwise closed at the end of test runs).
12. Check the **Show SAPGUI during single runs** check box to display the R/3 client during test runs.

    GUI display is only an option for Try Script executions. This setting is ignored for baseline tests and load tests.
13. Check the **Enable client-side scripting** check box to enable SAP client-side scripting via the registry.

    This option disables warnings that raise pop-up windows when new users start.
14. Check the **Use new SAP Visual Design** check box to enable SAP Visual Design for the SAP client.
15. Click the **Logging** tab to specify additional log levels.

    Note: These settings affect performance and resource utilization.
16. From the **Log level** list box, select a log level for virtual user logging:
    - `Disable` - Virtual user logging is disabled.
    - `Normal` - Default logging.
    - `Debug` - Detailed information is logged, such as additional session information and errors.
17. Check the **Capture screenshots** check box to generate screenshots for each activated window.

    Note: Screenshot capture is only enabled for Try Script executions. This setting is ignored for baseline tests and load tests.
18. Check the **Capture screenshots for every action** check box to have each action performed by the script documented with a screenshot.

    Note: Screenshot capture is only enabled for Try Script executions. This setting is ignored for baseline tests and load tests.
19. Check the **Log control information in TrueLog** check box to have additional control information written to the TrueLog.
20. Check the **Log control information for single run in TrueLog** check box to have additional control information written to the TrueLog only for try script and verification tests.
21. Check the **Log control information on error** check box to enable writing of control information to TrueLogs in the case of errors.

Note:

For large load-tests, it is also recommended that you uncheck **Log control information in TrueLog** and only select **Log control information on error**. These settings greatly improve overall performance, as not all information on each screen is logged. In the case of errors, control state is logged on the nodes that cause the errors; this allows you to troubleshoot problems, as you can see the full state of the active window.

22. Check the **Highlight controls** check box to have controls highlighted as they are accessed by the script.
23. Click the **Measuring** tab to specify a custom measurement level.

    By default, all available performance metrics are collected during test runs.
24. Check the **Enable all timers and counters for all controls** check box to enable all timers and counters for all actions on controls during replay.Or, select a specific control from the **Control** list box and select the specific counter types that you want to have applied to that control:

    - `Count round-trips` - The number of round-trips for the action on the control.
    - `Count response time` - The time it takes a function to return requested data.
    - `Count Interpretation time` - The time it takes to interpret a request.
    - `Count flushes` - The number of flushes per action.

    Note: Alternatively, you can click the **Apply to All Controls** button to apply your counter settings to all controls.
25. Click **OK** to save your settings.

Configuring Citrix XenApp Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, scroll down to and click the **Citrix** icon. The **General** tab opens.
4. In the **Synchronization timeout** text box, type a default synchronization timeout in milliseconds that is to be used by all `CitrixWaitForXXX` functions that do not specify a timeout value.

   The default value is `60000 ms`.
5. Check **Force window position** to make the calls to `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` move windows to the coordinates captured during recording (enabled by default).

   When this check box is not checked, both the `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` functions provide the parameter `bForcePos` to enable this option for each individual call.
6. Check **Disconnect on transaction end** to disconnect the Citrix client following the end of each transaction, including the `TInit` transaction (disabled by default).
7. Check **Gracefully disconnect session** to perform a log-off when disconnecting from a Citrix XenApp session (enabled by default).
8. Check **Log screen before each user action** to capture and write a screenshot at the beginning of each user action (enabled by default).

   When TrueLog generation is enabled, a screenshot is taken at the end of each synchronization function and written to the TrueLog.

The `CitrixWaitForScreen` function captures a screen region and checks it against a specified condition instead of against a hash value that is captured at recording. This file can later be examined or compared to what was captured during recording for error analysis.

If the function call `CitrixWaitForScreen` fails and **Dump window region on unsuccessful screen synchronization** is checked , the captured screen region is written to a file in the result directory. The function call `CitrixWaitForScreen` may fail, for example, if the condition does not match when the timeout period expires.

9. Check **Use RAM disk** to use a different drive for the intermediate storage of images.

   Note: If you check **Use RAM disk**, you must select the appropriate drive letter of the RAM disk from the list box. TrueLog generation performance improves if the specified drive is a RAM disk.
10. Select the **Simulation** tab.
11. In the **Length of time mouse button remains pressed** field, type the length of time that the virtual user is to hold the mouse button in the pressed state.

    The default value is `200 ms`. The functions `CitrixMouseClick` and `CitrixMouseDblClick` use this value.
12. In the **Length of time between the clicks of a double-click** field, type the maximum length of time that can pass between the two clicks of a double-click.

    The default value is `100 ms`. The function `CitrixMouseDblClick` uses this value.
13. In the **Mouse speed** field, type the speed (in pixels per second) at which the mouse is to move across the screen.

    The default value is `1000 pixels`.
14. In the **Length of time each key remains pressed** field, type the length of time that the virtual user must hold a keyboard key in the down state.

    The default value is `50 ms`. The functions `CitrixKey` and `CitrixKeyString` use this value.
15. In the **Length of time between keystrokes when entering strings** field, type the length of time that must pass between the individual keystrokes.

    The default value is `100 ms`. The function `CitrixKeyString` uses this value.
16. In the **Key repeat time** field, type the time required for a complete key stroke when simulating repeat functionality.

    The default value is `50 ms`. A value of `50 ms` represents 20 keys per second.
17. In the **Delay after successful synchronization** field in the **Think times** area of the tab, type the length of time that virtual users are to remain inactive after passing a successful synchronization point.

    The default value is `1000 ms`. The function `CitrixWaitForXXX` uses this value.
18. In the **Delay after each user action** field, type the length of time that virtual users remain inactive between actions.

    The default value is `100 ms`.
19. Click the **Citrix client** tab to specify Citrix client options.
20. From the **Network protocol** drop box, select the low-level network protocol to use for locating and connecting to the Citrix server.

    For more information, refer to Citrix client documentation.
21. Check the **Use data compression** check box to compress all transferred data (enabled by default).

    This feature reduces file size but requires additional processor resources.
22. Check the **Use disk cache for bitmaps** check box to store commonly used graphical objects, such as bitmaps, in a local disk cache (disabled by default).
23. Check the **Queue mouse movements and keystrokes** check box to queue mouse and keyboard updates (disabled by default).

    This feature reduces the number of network packets sent from the Citrix client to the Citrix XenApp server.

24. From the **SpeedScreen latency reduction** drop box, select one of the following to enhance user experience on slower network connections:

- For WANs and other slower connections, select `On`.
- For LANs and other faster connections, select `Off`.
- To turn latency reduction on and off based on the latency of the connections, select `Auto`.

25. From the **Encryption level** drop box, select a level of encryption for the ICA connection.

    The Citrix XenApp server must be configured to allow the selected encryption level or higher.

    Note: Using an encryption level other than the server default or `Basic` disables automatic logon to the Citrix XenApp server.

26. Click **OK** to save your settings.

Setting .NET Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **.NET** icon.
4. Check the **Route HTTP .NET through** Silk Performer **web engine** check box to route all HTTP/HTTPS traffic that is generated by your .NET test driver through the Silk Performer Web engine.

   This allows you to take advantage of the features that Silk Performer offers for accurate testing, such as its high performance Web engine, modem simulation, detailed timing/throughput information for each transmission, HTTP/XML TrueLog, and IP spoofing.

   The **Routed web service proxy classes** text box displays the .NET Web Service client classes that are used in your .NET test driver. HTTP/SOAP traffic of selected classes is routed through Silk Performer's Web engine, regardless of the setting above. This setting can only be changed via the Silk Performer .NET AddIn.
5. Click the **One application domain for each virtual user container process** option button to share one .NET application domain among all virtual users running in a virtual user process.

   This minimizes the administrative overhead for the .NET Common Language Runtime, but may cause runtime problems because all objects will be loaded into the same application domain.
6. Click the **One application domain for each virtual user** option button to give each virtual user its own .NET application domain.

   This increases the administrative overhead of the .NET Common Language Runtime, but ensures that objects from different virtual users will not interfere with one another.
7. Click **OK** to save your settings.

CORBA/IIOP Settings

Setting CORBA/IIOP Replay Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **CORBA/IIOP** icon.
4. Click the **Replay** tab.
5. Use the **IIOP 1.2** area to specify the default settings for IIOP 1.2 replay.
6. In the **Max. chunk size for value type** text box, type the maximum byte size for a single part of a chunked-encoded value type.

   A chunked-encoded value type is a value type that has been broken into smaller parts
7. Check the **Use fragmentation** check box to fragment IIOP request and response messages when they exceed the length set in the **Fragmentation size** text box.
8. In the **Fragmentation size** text box, set the fragmentation byte size for a chunked-encoded value type.

   When a request or response message exceeds this size, fragmentation is used to send the message.
9. Click **OK** to save your settings.

Setting CORBA/IIOP WChar/WString Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **CORBA/IIOP** icon.
4. Click the **WChar / WString** tab.
5. Use the **Code set** area to specify the settings for the wide-character code set.

   In general, a code set determines the encoding rules for characters and wide characters. Code sets can be byte-oriented or non-byte-oriented (for example, ASCII is a byte-oriented code set for single byte characters). Multibyte characters defined by non-byte-oriented code sets have a byte-orientation (big-endian, little-endian); byte-oriented code sets do not.
6. Check the **Unicode** check box to use the Unicode code set.

   Unicode is a non-byte-oriented code set for two-byte characters. The byte orientation of Unicode characters is determined by the IIOP message byte order.
7. From the **WChar size** list box, select the size of a CORBA wide character (wchar).
8. Click the **Byte-oriented** option button to have a byte-oriented code set used for CORBA wchars and wstrings.

   Alternatively, click the **Non-byte-oriented** option button to use a non-byte-oriented code set for CORBA wchars and wstrings. If you select this option, use the **Script byte order** list box to select the byte orientation of non-byte-oriented CORBA wchar/wstring representation in the BDL script. The byte orientation can be `little-endian` (least significant bit first) or `big-endian` (most significant bit first).

   Note: Use the **IIOP 1.2** area to specify settings for IIOP 1.2. The wchar and wstring encoding in IIOP 1.2 differs from the encoding in previous IIOP versions. However, not all ORBs use the correct encoding for IIOP 1.2.
9. Check the **CORBA compliant** check box if the ORB uses CORBA compliant wchar/wstring encoding in IIOP 1.2.
10. In the **WChar** area, check the **No length encoding** check box to use pre IIOP 1.2 encoding.

    CORBA compliant IIOP 1.2 encoded wchars use a length indication for the size of the wchar.
11. In the **WString** area, check the **Null-terminated** check box if the ORB uses null-terminated wstrings.

    CORBA compliant IIOP 1.2 encoded wstrings are not null-terminated.
12. Click the **Chars** option button if the ORB uses the number of chars as wstring length.

    CORBA compliant IIOP 1.2 encoded wstrings specify the string length in bytes (chars).

    Alternatively, click the **WChars** option button if the ORB uses the number of wchars as wstring length.

13. Click **OK** to save your settings.

Setting CORBA/IIOP Recording Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **CORBA/IIOP** icon.
5. Click the **Recording** tab.
6. Use the **Proprietary ORB features support** area to specify whether to consider ORB-specific features when generating test scripts based on recorded IIOP traffic.

   When appropriate options are selected, generated scripts become more readable.
7. Select IIOP traffic-recording options:

   - **Orbix** for Orbix-specific features
   - **OrbixWeb** for OrbixWeb-specific features
   - **VisiBroker C++** for VisiBroker for C++ features
   - **VisiBroker Java** for VisiBroker for Java features
   - **M3 C++** for BEA Millennium for C++ features
   - **M3 Java** for BEA Millennium for Java features
8. From the **CORBA version of IDL file** list box, select the CORBA version of the application under test.

   When an application uses a CORBA version between 2.0 and 2.2, select `2.0` from the list box. When an application uses CORBA 2.3 or CORBA 3.0, select `3.0` from the list box.
9. In the **Preprocessor arguments** text box, enter any number of arguments that are to be passed to the preprocessor of the IDF file parser.

   For example, the arguments may be used to specify the include directory and to execute predefined macros.
10. Use the **IDL files loaded** list to view all of the IDL files that are currently used by the Silk Performer Recorder to generate high-level test scripts.

    Whenever the corresponding interface is defined in an IDL file, the Recorder generates type-dependent functions for specifying the parameters of CORBA operation calls in the test script.
11. Click **OK** to save your settings.

Java Settings

Configure Java settings, including classpath, version, JVM, JIT compiler, and logging options.

Configuring Java Version and Classpath Settings

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Java** icon. The **General** page opens.
3. Specify the directory of the Java home path in the **Java 32-bit home** or **Java 64-bit home** field, depending on what Java architecture you use. You can switch between 32-bit and 64-bit on the **Advanced** tab.

   This option enables the Java Virtual Machine to be loaded from a different path than is specified in the `PATH` environment so that you can switch between various Java Virtual Machines without changing the system `PATH` environment.

Note: Silk Performer automatically checks the path you specify here. If the path is not correct, the default Java home path of the operating system is used instead.

The **Classpath** specified for the Java Virtual Machine displays. By default the classpath is set to the system classpath.

4. Click **Check JVM** to verify your Java Virtual Machine configuration settings.

   To ensure that the Java environment is set up properly, the configuration should be tested.
5. Click **New File** to navigate to a class file and add it to the classpath of the Java Virtual Machine.

   Note: Press `Ctrl` or `Shift` to select multiple class files and add them to the classpath.
6. Click **New Directory** to navigate to a directory and add it to the classpath of the Java Virtual Machine.
7. To move selected files up or down in the classpath hierarchy, click **Move Item Up** or **Move Item Down**.
8. To delete a selected file from the classpath, click **Delete**.
9. Click **OK** to save your settings.

Setting Advanced Java Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Java** icon.
4. Click the **Advanced** tab.

   Use the **Advanced** area to set advanced options for the Java Virtual Machine.
5. In the **Command line options** text box, type any command-line options to pass to the Java Virtual Machine.
6. Check the **Disable JIT compiler** check box to disable the Just In Time (JIT) compiler of the Java Virtual Machine.
7. In the **JVM dll** text box, type the name of any Dynamic Link Library (DLL) that implements the Java Virtual Machine that is to be used.

   If necessary, click **(...)** to navigate to and select the Dynamic Link Library to use.
8. Click **Check JVM** to verify your Java Virtual Machine configuration settings.
9. Click **OK**.

Setting Java Logging Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Java** icon.
5. Click the **Logging** tab.

   Use the **Record settings** area to specify that additional properties have their values logged during recording.

6. Set the log level for virtual user logging from the **Log level** list box.

   Note: To disable logging, set the log level to `None`.

   The following log levels are available:

   - `None` – No additional logs will be written.
   - `Error` – Only errors that occur during replay will be logged.
   - `Normal` – Java messages will be logged.
   - `Debug` – Additional information will be logged (for example, events).
7. Click **OK** to save your settings.

Database Settings

Configure database settings, including SQL, ODBC, and recording options.

Setting SQLSTATE-specific Error Handling

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Database** icon.
4. Click the **SQLSTATE** tab.
5. Use the **Simulation exceptions** list to define exceptions for specific ODBC error codes to prevent termination of the simulation.

   When a database error occurs, Silk Performer automatically terminates the entire simulation. This list includes all the exceptions that Silk Performer uses for handling database-specific errors. The **Error** field lists the native error code. The **Class** field ranks the severity of the error. `Class 0` errors can be handled within a Silk Performer transaction. `Class 1` errors end the transaction but do not cause the simulation to terminate.
6. To add an error to the list of database-specific error exceptions, click the **Add** button. The **Edit Error** dialog box opens.
   a) In the **Error** field, add the error code for an error to be added to the exceptions list.
   b) In the **Class** field, add the error class.
   c) In the **Description** field, add a description for the error.
   d) Click **OK**.
7. To edit an error, select it from the list, and click **Edit**. The **Edit Error** dialog box opens.
   a) Edit the error code, class, and description as required.
   b) Click **OK**.
8. To remove an error from the exceptions list, select it and click the **Remove** button.
   a) Click **Yes** on the deletion confirmation dialog.
9. On the **Profile - [<profile name>]** dialog box, click **OK** to save your changes.

Setting ODBC-specific Error Handling

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **Database** icon.
4. Click the **ODBC Database Errors** tab.
5. Use the **Error definition** list to define exceptions for specific database error codes in order to prevent the termination of simulations.
6. To add an error to the list of database-specific error exceptions, click the **Add** button. The **Edit Error** dialog box opens.
   a) In the **Error** field, add the error code for an error that is to be added to the exceptions list.
   b) In the **Class** field, add the error class.
   c) In the **Description** field, add a description for the error.
   d) Click **OK**.
7. To edit an error, select it from the list, and click **Edit**. The **Edit Error** dialog box opens.
   a) Edit the error code, class, and description as required.
   b) Click **OK**.
8. To remove an error from the exceptions list, select it and click the **Remove** button.
   a) Click **Yes** on the deletion confirmation dialog.
9. On the **Profile - [<profile name>]** dialog box, click **OK** to save your changes.

Setting High-Level ODBC Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Database** icon.
4. Click the **ODBC High-Level** tab.
5. Use the **Database** area to specify details about the ODBC data source you will use and how Silk Performer should interact with the ODBC driver.
6. From the **Data Source Name** list box of available ODBC data sources, select the data source that you want to connect to for testing.

   Note: To set up an additional ODBC data source, use `Windows 32-bit ODBC Administrator` (via the Windows Control Panel).

   The data source you select here is used only if you use high-level ODBC functions that perform an implicit connection to the database. When you use medium-level ODBC functions, this data source is not used. Instead, the connection is performed with an explicit `DB_Connect("connection string")` function in the script.
7. In the **UserID** text box, enter the user name that is to be used to connect to the data source.

   You must specify a valid user for the DBMS to which you want to connect. If you use a trusted connection to connect to your DBMS, leave this field blank. Note that the user ID you enter here will only be used if you use high-level ODBC functions.
8. In the **Password** text box, enter the password that is to be used when connecting to the data source.

   You must specify a valid password for the database user you are connecting. If you use a trusted connection, or if the user has no password, leave this field blank. Note that the password you enter here, as well as all remaining options you choose in this area, are used only if you use high-level ODBC functions.

9. From the **Isolation** list box, select the level of isolation for the entire test.

   For a detailed description of isolation levels, consult your DBMS documentation and see the ODBC functions reference. You can change the isolation level and any other DBMS specific settings using the ODBC functions `SQLSetConnectOption` and `SQLSetStmtOption` in your script.
10. From the **SQL concurrency** list box, select a type of cursor concurrency to use for the scrollable cursors in your test script.

    For a detailed description of concurrency settings, consult your DBMS documentation.
11. From the **SQL cursor type** list box, select a cursor type for the scrollable cursors in your test script.

    For a detailed description of cursor type settings, consult your DBMS documentation.
12. Check the **Autocommit** check box to commit the database automatically after each SQL command.

    Otherwise the database will be committed only after an explicit commit statement. This setting is effective for all connected cursors.
13. Check the **Reprepare SQL commands** check box to prepare and execute a SQL statement each time it is called.

    Otherwise Silk Performer prepares SQL statements only when necessary. This setting is effective for all cursors used in your test.
14. Click **OK** to save your settings.

Setting Database Recording Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Database** icon.
5. Click the **Recording** tab.
6. In the **Oracle OCI** area, click the **Generate random input for long placeholders** option button to replace large amounts of data exchanged between the client application and the database server with random data.

   Because original data is rarely required for replay, this option helps to prevent large data files from being carried along with test scripts.
7. Click the **Generate files for long placeholders** option button to store large amounts of data exchanged between the client application and the database server in a file.

   During replay, the file is used to send exactly the same data to the database server that is sent to the client application.
8. Click **OK** to save your changes.

Test Scripts

Test scripts prescribe the actions of the virtual users that are run during tests. Scripts are written in Silk Performer's proprietary scripting language, the Benchmark Description Language (BDL). The usual way to create scripts is to have the Silk Performer Recorder generate them from real network traffic that has been captured and recorded. Scripts can also be generated manually, or sample scripts provided by Silk Performer can be modified for reuse.

As many test scripts as necessary can be added to a project, and existing scripts can be removed. Scripts need to be compiled before they can be used. Individual scripts in a project can be compiled, or all the scripts associated with a particular project can be recompiled at once.

Adding Test Scripts

1. In the **Project** menu tree, right-click the **Project** node and choose **Add Existing Script**.

   Tip: Alternatively, choose **Project** > **Add Existing Script** from the menu bar.

   The **Select Script(s)** dialog box opens.
2. Navigate to and click the script to add to the project.
3. Click **OK** to save your settings.

Compiling Test Scripts

1. In the **Project** menu tree, expand the **Scripts** node.
2. Right-click the script you want to compile.
3. Select **Compile Script** from the context menu.

   Alternatively, select **Script** > **Compile** from the menu bar.

   You can monitor the results of the compilation below in the **Compiler** page.

Removing Test Scripts

1. In the **Project** menu tree, expand the folder of the current project.
2. Expand the **Scripts** folder.
3. Right-click the script you want to remove.
4. Select **Remove Script** from the context menu. You can optionally select **Remove all Scripts** to remove all scripts from your project.
5. Click **Yes** on the deletion confirmation dialog.

Data Files

Data files that are required by Silk Performer scripts [user-data files (`CSV`), `IDL`, certificate files (`PEM`), random data files (`RND`), and type library files (`TLB`, `DLL`, `EXE`, `OCX`, `OLB`, `PKG`)] are added to projects by specifying the directory where they are located. The compiler first searches for random data files in the directory where the `BDF` file is located, then in the specified directory. If a random data file from a different location needs to be used, the full path to the file can be specified in the script file.

Note: It is important that all the user data files needed by your script are added to your project. Otherwise they will not be available to remote agent computers during tests.

Adding Data Files

1. In the **Project** menu tree, right-click the **Project** node and choose **Add Data File**.

   Alternatively, you can choose **Project** > **Add Data File** from the menu bar.

   The **Select Data File(s)** dialog box opens.
2. Use the **Select Data File(s)** dialog box to navigate to and select the data file to add to the project.
3. *Optional:* Select data files for addition by inserting wildcard characters into the **File name** text box.

   Note: If you insert wildcard characters into the **File name** text box, you must also click **Add with wildcards** to select the files that match the specified characters and add them to your project's data files.

4. *Optional:* Check the **Include subfolders** check box to add files from subfolders that match the specified wildcard characters.
5. Click **OK** to save your settings.

Removing Data Files

1. In the **Project** menu tree, expand the current project.
2. Expand the **Data Files** folder.
3. Right-click the data file you want to remove.
4. Select **Remove Data File** from the context menu.
5. Click **Yes** on the deletion confirmation dialog.

Compressing Data Files

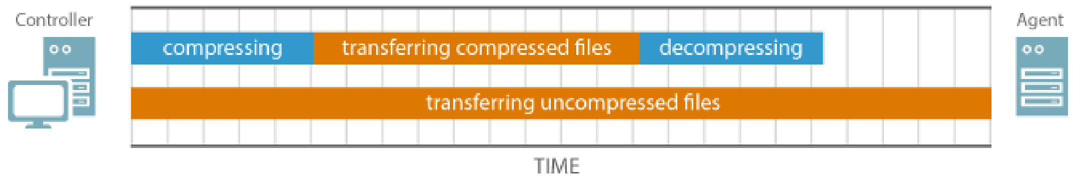
In some scenarios you can speed up initializing your load test by compressing the data files prior to transferring them to the agent computers. To compress data files, click **Settings** > **System** in the Silk Performer menu, click the **Control** tab, and enable the check boxes **Compress data files for LAN/WAN agents** and **Compress data files for cloud agents**.

Compression of data files for cloud agents is enabled by default.

Compressing data files can be especially useful ...

- when you are using network connections with a small bandwidth. When your agents reside in a local area network (LAN), compressing data files is not as useful, because local area networks are usually fast. If, however, your agents reside in a wide area network (WAN), deploying compressed data files can save a lot of time.
- when you are using many agents for your load test. Compressing data files is done only once (on the controller), decompressing data files is done once on each agent. The decompression process is done in parallel, so it does not necessarily slow down initializing your load test. Nevertheless: the more agents you use, the more likely it is that you save time with compressed data files.
- when you are using data files that are well compressible. Text files or .csv files are examples for well compressible files. These can be shrinked considerably and compressing such files is done rapidly.

Whether compressing data files is useful, depends on your very specific environment and load test. As a general rule: If compressing, transferring the smaller data files, and decompressing files takes less time than just transferring the uncompressed files, then data file compression is useful.


Controller
compressing
transferring compressed files
decompressing
transferring uncompressed files
Agent
TIME

Caching Data Files

Before a load test is executed, it first needs to be initialized. Part of the initialization is the deployment of data files from the controller to the agents. Deploying data files can take some time, depending on their number and size. To speed up initialization, you can instruct Silk Performer to cache data files on the agents. In this case, data files must be deployed only for the first load test. Subsequent load tests will then be initialized quicker.

To cache data files, click **Settings** > **System** in the Silk Performer menu, click the **Control** tab, and enable the check box **Cache data files on agents**.

When caching is disabled, data files are transferred to the remote agent and stored in the remote project folder. When caching is enabled, the data files are stored in `<public documents>\Silk Performer`

`<version>\DataCache`. The `GetDataFilePath` function provides the correct data file path in either case. Do not use the function `GetDirectory` with `DIRECTORY_PROJECT` to locate data files, as it does not work when data file caching is enabled.

Agent Computers

Agent computers host the virtual users that are run during tests. As many agent computers as necessary can be added to a project to support the required number of virtual users.

Agent computers are assigned to particular projects from the pool of agent computers that are available to the controller computer. The maximum number of virtual users who will run on each agent is computed automatically by Silk Performer; Alternatively, this number can be set manually. New agents can be added to the agent pool from the local area network or from other Windows domains; they are then available to be added to projects. Agents can be checked for availability, and agents can be removed from the pool.

Configuring Project Agents

1. On the **Project** menu tree, right-click the **Project** node and choose **Configure Project Agents**. The **Configure Project Agents** dialog box opens.

   Note: Place your cursor over an agent name to view that agent's system and agent-version details.

   Tip: Alternatively, click **Assign Agents** on the Silk Performer workflow bar. Then click the **Configure Project Agents** link on the **Setup Agents** dialog box.
2. Double-click an agent computer in the **Available Agents** pane to add the agent to the project.

   Note: It is not possible to remove all agents from the **Project Agents** pane. `Localhost` is added by default when you leave the list empty.
3. To add a foreign agent (indicated by a question mark icon) from the **Project Agents** pane to the agent pool, right click the agent and choose **Add to Agent Pool**.

   Foreign agents are agents that have been configured for a project that have not been added to the agent pool.

   a) To make a newly added agent available for assignment, click **Configure Agent Pool** and configure the agent on the **System Settings - Agents** page.
4. To add an agent that does not appear in the **Available Agents** list to the agent pool, click **Configure Agent Pool** to go to the **System Settings - Agents** page.
5. Click **OK**.

Removing Agent Computers

1. In the **Project** menu tree, expand the current project.
2. Expand the **Agents** folder.
3. Right-click the agent computer you want to remove.
4. Select **Remove Agent** from the context menu.

   Note that you must leave at least one active agent in your project.
5. Click **Yes** on the deletion confirmation dialog.

Modeling Scripts

Before you can conduct a Silk Performer load test you need to create a test script that prescribes the actions of the simulated users run during the test. The script is written in Silk Performer's proprietary scripting language, the Benchmark Description Language (BDL).

Scripts can be created in different ways depending on the application type. The standard (and typically easiest) method for creating a test script is to use the Silk Performer Recorder to capture and record traffic that is representative of the type you need to simulate in your test. The Silk Performer Recorder automatically generates a BDL test script based on the recorded traffic.

Another method of creating a test script is to manually create a new script in BDL. A variant on the manual approach is to create a test script based on one of the sample BDL scripts that are provided by Silk Performer.

Recorded Test Scripts

The standard, and easiest, method of creating a test script is to use the Silk Performer Recorder, Silk Performer's engine for capturing and recording traffic and generating test scripts.

First, the Silk Performer Recorder captures and records a representative amount of real traffic between a client application and the server to be tested. When recording is complete, the Silk Performer Recorder generates either a test script or a capture file. You can create scripts out of the capture file later on. The script is written in Silk Performer's proprietary scripting language, the Benchmark Description Language (BDL).

During the recording phase, you define transactions. A transaction is a discrete piece of work that will later be assigned to a virtual user in a load test and for which separate time measurements will be made. You should create a new transaction only for a piece of work that has no dependencies on another piece of work. Individual time measurements can be made for any action or series of actions that happen during recording.

Using the Silk Performer Recorder has a number of advantages:

- Recorded traffic and function calls include a lot of static information. Replay of static information ordinarily does not lead to reliable benchmark results. Because recorded scripts can easily be customized, you can add random functions to generate dynamic and realistic workloads .
- Transactions defined during recording modularize the script into clearly laid out sections. It is easy to modify workloads based on different statistics or requirements using the randomization functions embedded in Silk Performer.
- Timers set during recording provide user-defined granularity for measuring any part of a transaction.
- Development time and costs can be reduced considerably. There is no need to change the front-end application to simulate new behaviors. It is sufficient to modify the script and repeat the simulation until the requirements are met. The front-end application has to be implemented only once; the script then becomes its prototype and reproduces its behavior.

Recording a Test Script

If you are an experienced user, you may want to create a script on your own and write your code manually. You can start with a blank script or you can customize one of the preinstalled sample scripts.

However, you can also use the Silk Performer Recorder, which does the scripting and the recording work for you. Here are the basic steps you need to perform when you use the Silk Performer Recorder:

1. Open the project in which you want to work, or start a new project.
2. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
3. From the **Recording Profile** list, select the profile for the client application that you plan to test. If a profile has not yet been set up for the application you want to use, click **Settings** to the right of the list to set one up.
4. Depending on the project type, enter either the application's URL in the **URL** field or the location of the application in the **Command line** field.
5. Click **Start recording**. The Silk Performer Recorder dialog opens in minimized form, and the client application starts.
6. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.

7. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.
8. Insert transactions and timers into the test script during the recording phase. You can create as many transactions and timers as you want. To insert a transaction, click the **New Transaction** button. A transaction represents a piece of work that can be assigned to a virtual user.
9. In the ensuing dialog, enter a name for the transaction and click **OK**. The new transaction appears in the **Actions** log.
10. To insert a timer, click the **New Timer Session** button. A timer is a user-defined measurement period in a test. You should create timers for each component of a transaction for which you want to analyze performance. In the ensuing dialog, enter a name for the timer and click **OK**.
11. To end recording, click the **Stop Recording** button.
12. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.
13. Close the client application and close the Silk Performer Recorder.

Secure Connections and Certificates

If your application under test is accessed over a secure connection, Silk Performer needs to be configured so that the communication between the client and the server is trusted. This is done with certificates, where you can chose between the following two approaches:

Micro Focus Certificate Authority (CA) Certificate

The Micro Focus CA certificate, located at `C:\Program Files\Silk\Silk Performer 20.0\IRCAcert.crt`, needs to be installed on the client to establish a secure connection between the client and the server. The according server certificate, issued by the Micro Focus Certificate Authority, is located at `C:\Program Files\Silk\Silk Performer 20.0\IRServerCert.pem`.

Connection route:

1. The client opens a secure connection to the Recorder, which acts as a server.
2. The Recorder sends back the Micro Focus server certificate.
3. The client receives the certificate and checks whether it was issued by one of the Trusted Root Authorities (Micro Focus CA certificate). If the check is successful, the secure connection is established.
4. The Recorder tries to open a second secure connection to the system under test.
5. The system under test sends back its own server certificate.
6. The Recorder accepts this certificate and establishes a secure connection to the system under test.

Server Certificates

If you have the server certificate of your system under test, you can specify it in the **Profile Settings** to enable secure recording and replay.

Connection route:

1. The client opens a secure connection to the Recorder and receives the server certificate that you specified in the **Profile Settings**, so the client thinks it is already connected to the system under test.
2. The Recorder opens a second secure connection to the system under test.
3. The system under test sends back its own server certificate.
4. The Recorder accepts this certificate and establishes a connection to the system under test.

Native Mobile Apps

Native apps on mobile devices that communicate over a secure connection often match received server certificates (either the server certificate from the system under test or the Micro Focus server certificate) against a specific string, for example a company or domain name. This security approach prevents the Micro Focus certificate approach from working.

Generating Scripts from Capture Files

What is a capture file?

While recording a user transaction, the Silk Performer recorder creates a so-called Silk Performer capture file, which contains the entire traffic of the recorded session. When you stop the recording, Silk Performer saves the capture file, analyzes the traffic in the background, and displays the **Capture File** page. On the **Capture File** page you can adjust a variety of settings, like resolving potential problems, applying filters, or setting recording rules. When you are done, click **Generate Script** to generate a .bdf file out of the .spcap file.

Tip: You can click the check box **Always show Capture File page after recording**, which displays on the bottom right of the **Capture File** page. If you disable this check box, Silk Performer will not show the **Capture File** page and will automatically create a script out of the capture file.

Tip: It is possible to generate several scripts (with different options) from the same capture file.

Which project types use capture files?

Capture files will be created for the following project types:

- `Web business transaction (HTML/HTTP)`
- `Web (Async)`
- `Web low level (HTTP)`
- `Mobile Devices`
- `Flex/AMF3 (Adobe)`
- `Flex/AMF3 (GraniteDS)`
- `Silverlight`
- `HTTP Live Streaming (HLS)`
- `Java over HTTP`
- `WebDAV (MS Outlook Web Access)`
- `Oracle ADF`
- `Email (SMTP/POP)`
- `Directory server (LDAP)`
- `Radius`
- `FTP`
- `TCP/IP based application`
- `Mixed Protocols`

Specifics for Recording Flex/AMF3 and Java over HTTP

When you work with the project types `Flex/AMF3 (Adobe)`, `Flex/AMF3 (GraniteDS)`, or `Java over HTTP`, the following specifics apply:

When the **Capture File** page displays issues (which can occur during creation of the capture file, due to missing .jar files for example), use the buttons in the **Resolve Problems** area to fix these. Once all problems are resolved, click **Generate Script**.

Traffic Filters

When you record a web application, the resulting script can include a lot of unwanted, application-independent traffic. This unwanted traffic can overload a script considerably and it can make a script look messy when it actually contains just a few lines of application-relevant code.

To tidy up a script and include only the traffic that was intended to be recorded, you can use the filters on the **Capture File** page. Silk Performer provides the following filters:

- A **general filter**: This filter allows you to exclude proprietary TCP/IP traffic, UDP traffic, and traffic from browser-specific domains.
- A **domain filter**: This filter allows you to exclude traffic from specific domains. You can use it to filter traffic from third-party components that are embedded into a website, like social media or statistic plugins.
- A **path and query filter**: This filter allows you to exclude web requests (API calls) from a script.

Filtering General Traffic from Capture Files

You can exclude proprietary TCP/IP traffic, UDP traffic, and browser-specific domains from your script by using the following filter on the **Capture File** page.

1. In the **Project** tree, expand the **Capture Files** node and double-click a capture file.
2. Enable **Exclude proprietary TCP/IP traffic**.

   Silk Performer automatically recognizes traffic of standard TCP/IP-based web protocols such as HTTP. Traffic from non-standard web protocols is often generated by the browser and can be described as proprietary TCP/IP traffic. It can be useful to exclude this traffic as it can clutter up a script and usually it is not the traffic that was intended to be recorded.
3. Enable **Exclude UDP traffic**.

   Browsers and web applications often use the UDP for device and service detection. UDP traffic is also usually not intended to be recorded.
4. Enable **Exclude browser-specific domains**

   When you record a web application, your script will usually also contain additional , browser-specific web calls. These web calls are independent of the recorded application and are only scripted when you use a certain browser to do the recording. Therefore, it is typically desirable to exclude these web calls, which helps to create a cleaner script. The tooltip shows which browser-specific domains Silk Performer found in your capture file.
5. Click **Generate Script**.

Filtering Domains from Capture Files

While recording a user transaction, the Silk Performer Recorder might also capture third-party components that are embedded into a website, like social media or statistic plugins. To remove this unwanted traffic, open a capture file, apply the domain filter, and generate a new script. The script will just contain the desired code from the user transaction.

1. In the **Project** tree, expand the **Capture Files** node and double-click a capture file.
2. To include the traffic from one specific domain, click **Include traffic from** and select a domain from the list box.
3. To include or exclude multiple domains, select one of the following options:

   - Click **Include only traffic from domains selected below** to preserve only the traffic from the below selected domains in the script.
   - Click **Exclude traffic from domains selected below** to leave out the traffic from the below selected domains in the script.
4. Expand the list of domains and select all domains that shall be included or excluded.

   Tip: To quickly drill into the list, right-click the nodes and click **Expand all** or **Collapse all**.
5. Click **Generate Script**.

The generated script will be modified as follows:

- Calls to filtered domains will be removed from the script.
- The BDL function `WebSetDomainSuppress` will be added to the script. Embedded objects are loaded automatically by page-level functions, even if they are not explicitly listed in the script. `WebSetDomainSuppress` avoids that.

Filtering Paths and Queries from Capture Files

You can exclude web requests (API calls) from a script by using the following filter on the **Capture File** page.

1. In the **Project** tree, expand the **Capture Files** node and double-click a capture file.
2. In the **Path and Query Filter Settings** enter one or more patterns. The matches in the script display immediately.
3. Click **Generate Script**.

The defined patterns are removed from the generated script.

Rule-Based Recording

This section explains how to configure Silk Performer's Recorder (HTTP, TCP/IP) using recording rule files. It describes the structure and syntax of recording rules and shows examples. Recording rules are an advanced Silk Performer concept. You need to have a thorough understanding of the involved protocols (TCP/IP, HTTP and HTML) and of Silk Performer.

Note: You can not only manually script recording rules, you can also use templates from the **Recording Rules** tab.

Recording rules allow you to configure the Recorder in a number of ways:

- TCP/IP: By providing protocol descriptions of proprietary TCP/IP based protocols.
- HTTP: By specifying the scenarios in which the Recorder should script parsing functions for dynamically changing values and generate replacements for those values.

Recording Rule Files

Recording rule files are written in XML and have the file extension .xrl. They contain the rules for the Silk Performer Recorder.

Project specific recording rules are stored in the documents directory of the respective project, for example: `<my documents>\Silk Performer <version>\Projects\TestProj\Documents\`

Global recording rule files are stored in the public or the user's RecordingRules directory, for example: `<public user documents>\Silk Performer <version>\RecordingRules\`

The root node for all recording rule file node trees is `RecordingRuleSet`. Here are the three types of recording rules with their respective XML node:

- HTTP parsing rules: `HttpParsingRule`
- TCP rules for WebTcpipRecvProto(Ex): `TcpRuleRecvProto`
- TCP rules for WebTcpipRecvUntil: `TcpRuleRecvUntil`

Writing XML Files

Consider the following encoding requirements when you write XML files:

Encoding Special Characters

Some characters and symbols must be encoded for a proper XML syntax:

Description	Character	XML representation
Less Than	<	&lt;
Greater Than	>	&gt;
Quote	"	&quot;
Ampersand	&	&amp;

Encoding Binary Data

Binary data must be encoded in hexadecimal notation:

Description	XML representation
CR (carriage return)	&#x0D;
LF (line feed)	&#x0A;
NULL-byte	&#x00;

Recording Rule File Example

Naming conventions used in this example

XML nodes are referred to by their full path names: `RecordingRuleSet\HttpParsingRule\Search\LB\RegExpr`

When base paths are clear, the relative paths of XML nodes are used: `Search\LB\RegExpr`

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordingRuleSet>
  <HttpParsingRule>
    <Name>Siebel Session Cookie</Name>
    ...
  </HttpParsingRule>
  <TcpRuleRecvProto>
    <Name>Siebel TCP Protocol</Name>
    ...
  </TcpRuleRecvProto>
  <TcpRuleRecvUntil>
    <Name>Telnet screen</Name>
    ...
  </TcpRuleRecvUntil>
</RecordingRuleSet>
```

Recording Rule Data Types

Recording rule attributes come in a variety of data types. The following table lists all valid data types and associated attribute descriptions:

Data Type	Description
Strings	All string values, including binary data (except NULL-Bytes), are valid.
Binary Data	All binary data, including NULL-Bytes, are valid.

Data Type	Description
Numbers	Unsigned numbers in the range of 0 thru 4294967295 are valid. The value 4294967295 is the Max unsigned number.
Signed Numbers	Signed numbers in the range of -2147483648 thru 2147483647 are valid. The value 2147483647 is the Max signed number.
Numeric Ranges	Numeric ranges are notated using the following syntax: `[ MinValue ] [ "-" ] [ MaxValue ]` `MinValue` and `MaxValue` are unsigned numbers. If `MinValue` is omitted, a default value of 0 is used. If `MaxValue` is omitted, a default value of Max unsigned is used.
Boolean Values	Boolean values are valid data types. These include *true* and *false*.
Extended Boolean Values	Extended Boolean values are valid data types. These include *true*, *unknown* and *false*.
Distinct Values	Many distinct values are valid. Valid values depend on specific attributes.
Distinct Value Lists	Comma-Separated Value (CSV) lists are valid. Valid values depend on specific attributes.
Structured Data	Compound data types consisting of nested XML nodes are valid.

General Attributes of Recording Rules

All recording rule types allow users to specify both `Name` and `Active` attributes.

Attribute	Description
`Name` attribute	Data type `strings` Default value: `Unnamed Rule` This attribute specifies a name for a given recording rule. Names do not require a special meaning or syntax. They may appear in recorded script comments.
`Active` attribute	Data type `Boolean Values` Default value: `true` This attribute specifies whether or not a given recording rule is active. Inactive recording rules are ignored. This attribute allows for the temporary disabling of recording rules without them being deleted from recording rule files.

HTTP Parsing Rules

HTTP parsing rules are specified by XML nodes with the name `HttpParsingRule`.

Purpose

HTTP parsing rules specify when the Recorder should generate the parsing function `WebParseDataBoundEx()` for dynamically changing values, and then substitute parsing results where appropriate. This enables the Recorder to directly generate working scripts-and thereby eliminates the need for visual script customization using TrueLog Explorer.

When recording HTTP traffic, the Recorder applies HTTP parsing rules with the following settings:

- Page-level Web API
- Low-level Web API with/without automatic page detection

The Recorder does not apply HTTP parsing rules when recording HTTP traffic with the TCP/IP-level API setting.

How HTTP Parsing Rule Application Works

HTTP parsing rule application involves two main steps:

- Finding possible replacements (called "Rule Hits" or simply "Hits").
- Scripting parsing functions and replacements.

Details regarding both steps can be specified in HTTP parsing rules.

Finding possible replacements

During recording, each server response is parsed for rule hits. HTTP parsing rules specify how parsing is to be done and which parsing results (hits) are to be retained for future use. Parsing functions are not generated in this step; hits are simply retained for future use.

Scripting Parsing Functions

When the Recorder scripts a string value (either a parameter of a function, or a form field value or name), it examines all identified hits and determines a set of hits that are non-overlapping substrings of the string that is to be scripted. The Recorder then scripts the necessary parsing functions and replacements, rather than scripting the original string.

Structuring HTTP Parsing Rules

To keep the two values separate, HTTP parsing rules consist of two sections named `Search` and `ScriptGen`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>

  <HttpParsingRule>
    <Name>Example Rule</Name>
    <Active>true</Active>
    <Search>
      ...
    </Search>
    <ScriptGen>
      ...
    </ScriptGen>
  </HttpParsingRule>
</RecordingRuleSet>
```

Conversion Functions

You can write your own custom conversion function and specify this conversion function in a recording rule. The conversion function is contained in a native DLL file (programmed in C/C++). Within the conversion function, the value in the parsing rule is converted using the specified function, while the parsing function only parses the original value.

For example, when values are returned by the server in a double format they are sent by the client in an integer format. Therefore, a simple parsing rule will not find any matches in the script. However by applying a custom conversion function that converts the integer to a double format, the parsing rule is able to find the expected matches again.

Note: A conversion function can only be specified for HTTP Parsing Rules.

Sample Recording Rule

Below is an example of a recording rule with the conversion function specified.

```
<RecordingRuleSet>
  <HttpParsingRule>
    <Name>Parse by Boundaries, Convert and Replace (with custom conversion
DLL)</Name>
    <Search>
      <SearchIn type="Select{Body|Header|All}">Body</SearchIn>
      <LB>
        <Str>&gt;</Str>
      </LB>
      <RB>
        <Str>&lt;/strong&gt;</Str>
      </RB>
      <Conversion>
        <Dll>SampleConversion.dll</Dll>
        <Function>ConvertToUpperCase</Function>
      </Conversion>
    </Search>
    <ScriptGen>
      <ReplaceIn type="MultiSelect{FormFieldValue|Url|PostedData}">FormFieldV
alue, Url, PostedData</ReplaceIn>
      <VarName>ParsedByBoundary</VarName>
      <GenDebugPrint>true</GenDebugPrint>
      <Conversion>
        <BdlFunction>MyConvertToUpperCase</BdlFunction>
      </Conversion>
    </ScriptGen>
  </HttpParsingRule>
</RecordingRuleSet>
```

Guided HTTP Parsing Rule Example

This section steps through the process of designing a HTTP parsing rule for the sample application, ShopIt V 6.0, which ships with Silk Performer.

Recording ShopIt V 6.0 Without Parsing Rules

The Silk Performer sample Web application, ShopIt V 6.0, was deliberately built in such a way that the Recorder has to script the context-less function `WebPageUrl()` with a form definition that contains a session ID. This was achieved by having JavaScript assemble an URL.

Recording ShopIt V 6.0 without recording rules results in a script with a hard coded session ID, as shown in the following example.

```
WebPageUrl(sParsedUrl, "Unnamed page", SHOPITV60_
KINDOFPAYMENT_ASP002);

// …
```

```
dclform
  SHOPITV60_KINDOFPAYMENT_ASP002:
    "choice" := "CreditCard",
    "price"  := "69.9",
    "sid"    := "793663905";
```

Customizing ShopIt V 6.0 With TrueLog Explorer

In executing a Try Script run, the hard-coded session ID causes a replay error. The session handling customization feature of TrueLog Explorer solves this problem, modifying the script as shown in the following example.

```
dclparam
  sSessionInfo1 : string;

dcluser
  user
    VUser
  transactions
    TInit : begin;
    TMain : 1;

var
  sFormSid1 : string;

// ...

  WebParseDataBoundEx(sSessionInfo1, STRING_COMPLETE,
    "name=\"", 3, "\"", WEB_FLAG_IGNORE_WHITE_SPACE
      | WEB_FLAG_CASE_SENSITIVE, 1);
  WebPageLink("Check out", "ShopIt - Check Out");// Link 3
  Print("sSessionInfo1: " + sSessionInfo1);

  sFormSid1 := sSessionInfo1;
  WebPageUrl("http://u2/ShopItV60/kindofpayment.asp",
    "Unnamed page", SHOPITV60_KINDOFPAYMENT_ASP003);

// ...

dclform
  SHOPITV60_KINDOFPAYMENT_ASP003:
    "choice" := "CreditCard",
    "price" := "15.9",

//  "sid" := "348363999";
    "sid" := sFormSid1;
```

A second Try Script run reveals that the customization was successful and that the script now runs correctly.

Transferring Customization Details to the Recorder

The script runs correctly now that it has been customized. However a problem exists in that every script that will be recorded in the future must be also customized.

HTTP parsing rules enable the Recorder to continue this type of customization automatically in the future- so that recorded scripts can be automatically generated without needing manual customization.

To do this, research must be done into how the session ID can be parsed. The customization offered by TrueLog Explorer offers a good place to begin. It reveals the API call where the session ID first occurs, and boundaries that can be used to parse the session ID.

Using TrueLog Explorer, the first occurrence of the session ID can be located in the HTML code, as shown in the following example.

```
<script LANGUAGE="JavaScript">
  function doProcess(mylink)
  {
    scheme="http://";
    server="u2";
    serverport="";
    path="/ShopItV60/";
    file="kindofpayment.asp?";
    name="348364005";
    price="15.9";
    choice="CreditCard";
    mylink.href=scheme + server + serverport + path
    + file + "choice=" + choice + "&price="
    + price + "&sid=" + name;
  }
</script>
```

The left boundary ("name=\"") and the right boundary ("\"") identified by TrueLog Explorer seem to be reasonable choices for parsing the session ID. Now an initial version of a HTTP parsing rule can be written for the Recorder.

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>
  <HttpParsingRule>
    <Name>ShopIt V6.0 Session Id</Name>
    <Search>
      <SearchIn>Body</SearchIn>
      <LB>
        <Str>name=&quot;</Str>
      </LB>
      <RB>
        <Str>&quot;</Str>
      </RB>
    </Search>
    <ScriptGen>
      <VarName>ShopItSessionId</VarName>
    </ScriptGen>
  </HttpParsingRule>
</RecordingRuleSet>
```

This rule file may be saved to the public RecordingRules directory of Silk Performer-so that the rule will be globally available to all projects. The file name doesn't matter, but the file extension ".xrl" must be used. Alternately, if the recording rule is to be used with only one project, the file may be saved to the Documents directory of a Silk Performer project.

ShopIt V 6.0 Session ID's don't appear in HTTP response headers, so it is specified that only response bodies are to be searched (using attribute `Search\SearchIn`).

The session ID can be found by searching a left boundary. This boundary is specified in the attribute `Search\LB\Str`. Note that the quote symbol must be encoded in XML using the character sequence "&quot;".

A single quote marks the end of session ID's. This is specified in the attribute `Search\RB\Str`. Here again, the quote character must be encoded.

Finally, specifics regarding how the variable for the parsing result should be named need to be defined. Names are specified using the attribute `ScriptGen\VarName`.

Try the Recording Rule

When the rule is used in a recording session, the result is a recorded script with lots of variables.

```
var
  gsShopItSessionId : string; // Confirm-Button
  gsShopItSessionId_001 : string; // 348364008
  gsShopItSessionId_002 : string; // myForm
  gsShopItSessionId_003 : string; // address
  gsShopItSessionId_004 : string; // city
  gsShopItSessionId_005 : string; // state
  gsShopItSessionId_006 : string; // zip
  gsShopItSessionId_007 : string; // ZipCode
  gsShopItSessionId_008 : string; // cardtype
  gsShopItSessionId_009 : string; // cardnumber
  gsShopItSessionId_010 : string; // expiration
  gsShopItSessionId_011 : string; // sid

MYFORM004:
  gsShopItSessionId_003 := "a",
  gsShopItSessionId_004 := "b",
  gsShopItSessionId_005 := "c",
  gsShopItSessionId_006 := "" <SUPPRESS> ,
  gsShopItSessionId_007 := "d",
  gsShopItSessionId_008 := "Visa",
  gsShopItSessionId_009 := "111-111-111",
  gsShopItSessionId_010 := "07.04",
  gsShopItSessionId_011 := "" <USE_HTML_VAL> ;
```

This is because the rule is too general. The boundaries specified don't simply apply to the parsing of the session ID, they apply to almost all of the form fields. Although this doesn't prevent the script from replaying successfully, it's overkill.

Create a More Specific Rule

A more specific rule that creates a parsing function only for the session ID is needed. There are several ways of achieving this.

The Recorder uses the boundary strings in the `Search` attribute of parsing rules to extract substrings from each HTTP response. These substrings are called "rule hits" or simple "hits." The Recorder remembers each hit. When scripting a string, the Recorder checks to see if any of the identified rule hits are included in the scripted string. If they are, the Recorder generates a parsing function for the rule hit and substitutes the resulting variable into the scripted strings.

Limit the number of rule hits

A more specific rule can be created based on the unique characteristics of ShopIt V 6.0 Session IDs.

Session ID properties to consider:

- They consist of digits only.
- Their length is always 9 digits.

Taking this into account, the rule can be extended.

Limit the number of rule hits by conditions

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>
```

```
  <HttpParsingRule>
    <Name>ShopIt V6.0 Session Id</Name>
      <Search>
        <SearchIn>Body</SearchIn>
        <LB>
          <Str>name=&quot;</Str>
        </LB>
        <RB>
          <Str>&quot;</Str>
        </RB>
        <CondRegExpr>[0-9]+</CondRegExpr>
        <CondResultLen>9-9</CondResultLen>
      </Search>
    <ScriptGen>
      <VarName>ShopItSessionId</VarName>
    </ScriptGen>
  </HttpParsingRule>
</RecordingRuleSet>
```

The attribute `Search\CondRegExpr` specifies a regular expression that is applied to each rule hit. Rule hits that do not match this regular expression are dropped. The regular expression in the example above specifies that only rule hits consisting of digits are relevant.

The attribute `Search\CondResultLen` specifies a range of acceptable length for rule hits. Example above specifies that only hits with exactly nine characters are relevant. A subsequent recording session using this modified rule is successful: The recorded script contains a parsing function for the session ID only.

Script recorded using a modified rule

```
var
  gsShopItSessionId : string; // 348364011

dclform
  SHOPITV60_KINDOFPAYMENT_ASP003:
  "choice" := "CreditCard",
  "price" := "15.9",
  "sid" := gsShopItSessionId; // value: "348364011"

MYFORM004:
  "address" := "a", // changed
  "city" := "b", // changed
  "state" := "c", // changed
  "zip" := "" <SUPPRESS> , // value: ""
  "ZipCode" := "d", // added
  "cardtype" := "Visa", // added
  "cardnumber" := "111-111-111", // changed
  "expiration" := "07.04", // changed
  "sid" := "" <USE_HTML_VAL> ;//value:"348364011"
```

Specify allowed usage of rule hits

Rather than limiting the number of rule hits, one can be more specific in specifying where in scripts rule hits are to be used. Consider the following for this example: The session ID occurs only in form field values where the form field name is "sid". By extending the `ScriptGen` section of the parsing rule (as shown in the example below), rule hits are used only under these specific criteria.

Be more specific in script generation

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordingRuleSet>
```

```
  <HttpParsingRule>
    <Name>ShopIt V6.0 Session Id</Name>
    <Search>
      <SearchIn>Body</SearchIn>
      <LB>
        <Str>name=&quot;</Str>
      </LB>
      <RB>
        <Str>&quot;</Str>
      </RB>
    </Search>
    <ScriptGen>
      <VarName>ShopItSessionId</VarName>
      <ReplaceIn>FormFieldValue</ReplaceIn>
      <Conditions>
        <CompareData>
          <Data>sid</Data>
          <ApplyTo>FormFieldName</ApplyTo>
        </CompareData>
      </Conditions>
    </ScriptGen>
  </HttpParsingRule>

</RecordingRuleSet>
```

The attribute `ScriptGen\ReplaceIn` specifies that rule hits may only be used when the Recorder scripts a form field value. The condition additionally specifies that a replacement is allowed only if the associated form field name is `sid`. Recording with this modified rule generates a script that is identical to the script generated using the original rule.

Creating a Conversion DLL

The conversion DLL has to be a native Win32 DLL (no .NET assembly). To create it in Microsoft Visual Studio, you have to create a Win32 project and select the application type DLL.

On 64-bit operating systems, Silk Performer uses a 64-bit process to analyze capture files and generate scripts. Thus, you have to build two versions of the conversion DLL: a 32-bit DLL, which is used for replaying scripts, and a 64-bit DLL, which is used for generating scripts. The 64-bit DLL has to be named `<name of the 32-bit DLL>_x64.dll` and located in the same directory as the 32-bit DLL.

If you cannot create a 64-bit DLL for any reason, you can force Silk Performer to use a 32-bit process for script generation by setting the following registry key to `1`: `HKEY_LOCAL_MACHINE\SOFTWARE\Silk \SilkPerformer\<version>\Force32BitCaptureAnalyzer`.

Attention: Using the 32-bit script generator can cause issues with large capture files.

Silk Performer has provided sample Microsoft Visual Studio projects with header and .cpp files that give you guidance on how to create your own custom conversion DLL. The samples are located at `<public user documents>\Silk Performer <version>\SampleApps\SampleConversion`.

After you have created the project, there are some additional project settings to change:

- The Character Set option has to be set to Use Multi-Byte Character Set (MBCS). This is a requirement, because the strings to be converted are passed as MBCS strings.
- It is recommended that you use the static version of the C-runtime to remove dependencies to C-runtime DLLs. These dependencies could cause problems if the C-runtime DLLs (of used version) are not installed on the agent or controller machine. To use the static C-runtime libraries, change the Runtime Library setting in C++ / Code Generation to the following:
  - Multi-threaded (/MT) in the Release configuration
  - Multi-threaded Debug (/MTd) in the Debug configuration

Recommendation

If you reference the conversion DLL in a recording rule, you have to copy the DLL either into the Silk Performer recording rules directory or into the project directory. When developing the conversion DLL in Microsoft Visual Studio, you could copy the DLL to one of these directories (recording rules or project) in a post-build step or change the output directory path in the Release configuration to the recording rules directory.

Exporting the conversion function

You must export the conversion function with the following signature:

```
extern "C"
{
  __declspec( dllexport )
  long MyConversionFunction(
  LPCSTR sOriginalValue,
  LPSTR sConvertedValue,
  unsigned long* psConvertedValueLen,
  void* pReserved);
}
```

The purpose of using C linkage (extern "C") is to turn off C++ name mangling of the exported function. If using C++ linkage, other information like the types of arguments would be put into the exported name of the function. Use the "pure" function name as the exported function name (this is `MyConversionFunction` in the example above).

Function arguments:

sOriginalValue	The MBCS string value that has to be converted.
sConvertedValue	The string buffer that receives the converted value as MBCS string.
psConvertedValueLen	Points to a variable which contains the length of the string buffer sConvertedValue. This variable’s value has to be set to the number of bytes written to sConvertedValue if the conversion succeeded. This variable value has to be set to the number of bytes required for the converted value if the size of the string buffer sConvertedValue is too small.
pReserved	Reserved for future usage.
Return value	• ERROR_SUCCESS (0): Returned if the conversion was successful. *psConvertedValueLen now contains the number of bytes written to sConvertedValue. • ERROR_INSUFFICIENT_BUFFER (122): Returned if the conversion failed because the size of of sConvertedValue (*psConvertedValueLen) is too small for the converted value. *psConvertedValueLen now contains the number of bytes required to store the converted value. The function will be called again with size of sConvertedValue increased to *psConvertedValueLen. • Return any other value to indicate that the conversion failed.

Section Search - Finding Rule Hits

Details about finding rule hits are specified in the Search section. All XML paths of properties described in the section are relative to `HttpParsingRule\Search`.

Introduction

Rule hits can be extracted from HTTP responses in two ways:

- by defining boundaries
- by applying a regular expression

Defining boundaries

When a rule defines boundaries, any occurrence of a left boundary within an HTTP response marks the beginning of a rule hit. From this point onward within the HTTP response, the first occurrence of a right boundary marks the end of the rule hit.

Left boundaries can be defined in three ways:

- Strings: Any occurrence of a given string in an HTTP response marks the beginning of a rule hit.
- Regular Expressions: Any substring of a HTTP response that matches a specified regular expression marks the beginning of a rule hit.
- Offset Calculations: HTTP responses are run through the Offset Calculation to determine the beginning of a rule hit. Offset Calculation is explained in section “Offset, Length”.

Right boundaries can be defined in four ways:

- Strings: The next occurrence of a given string after the left boundary position marks the end of the rule hit.
- Regular Expressions: The next sub string of the HTTP response matching the given regular expression after the left boundary position marks the end of the rule hit.
- Length: The end of a rule hit is determined by running part of an HTTP response (from the beginning of the rule hit through to the end of the response) through a Length Calculation. Length Calculation is explained in Section “Offset, Length”.
- Character type: The next character that matches a given set of character types marks the end of the rule hit.

Applying regular expressions

If a rule defines a regular expression, any substring of an HTTP response that matches that regular expression yields a rule hit. By default, the entire match is the rule hit. Alternately, the rule can define a tag number so that the tagged subexpression is the rule hit.

LB\Str Attribute

Value	Description
Type	Binary Data
Default	(empty)
Description	This attribute specifies a string for searching rule hits. Each occurrence of this string marks the beginning of a rule hit. When searching, the attributes `CaseSensitive` and `IgnoreWhiteSpaces` are used.

Examples

```
<LB>
```

```
  <Str>name=&quot;</Str>
</LB>

<LB>
  <Str>BV_EngineID=</Str>
</LB>
```

LB\RegExpr Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute specifies a regular expression for searching rule hits. Each substring of an HTTP response that matches the regular expression marks the beginning of a rule hit. When searching matching substrings, the attributes `CaseSensitive` and `IgnoreWhiteSpaces` are not used.

Examples

```
<LB>
  <RegExpr>name *= *[&quot;']</RegExpr>
</LB>
```

LB\Offset Attribute

Value	Description
Type	Signed Numbers
Default	(empty)
Description	This attribute specifies an offset value for an Offset/ Length calculation, as described in Section “Offset, Length”, Offset, Length. This calculation is applied to the entire HTTP response with the given offset and the length of 0. The beginning of the calculation's result marks the beginning of the rule hit.

Example

```
<LB>
  <Offset>17</Offset>
</LB>
```

RB\Str Attribute

Value	Description
Type	Binary Data
Default	(empty)
Description	This attribute specifies a string that searches for the end of rule hits. It is used with the beginning of each rule hit

Value	Description
	identified using `LB\Str`, `LB\RegExpr` or `LB\Offset`. The attributes `CaseSensitive` and `IgnoreWhiteSpaces` are used during searches.

Example

```
<RB>
  <Str>&quot;</Str>
</RB>
```

RB\CharType Attribute

Value	Description
Type	Distinct Value Lists
Default	(empty)
Description	This attribute specifies a list of character types and is used to search for the end of each rule hit found using `LB\Str`, `LB\RegExpr` or `LB\Offset`. The ends of rule hits are defined by the first character that matches one of the character types specified below.

Allowed values are:

- UpperCase: Uppercase characters
- LowerCase: Lowercase characters
- NewLine: Newline characters
- Digit: Digits 0-9
- HexDigit: Hexadecimal digits 0-9, a-z, A-Z
- Letter: letters a-z, A-Z
- White: Whitespaces
- WhiteNoSpace: Whitespaces, excluding blank spaces
- Printable: Printable characters
- NonPrintable: Non-printable characters
- EndOfString: End of Strings (single and double quote)
- NonBase64: Characters not used in Base 64 encoding

Example

```
<RB>
  <CharType>EndOfString, White</CharType>
</RB>
```

RB\RegExpr Attribute

Value	Description
Type	Strings
Default	(empty)

Value	Description
Description	This attribute specifies a regular expression that searches for the end of rule hits. It is used with the beginning of rule hits found using `LB\Str`, `LB\RegExpr` or `LB \Offset`. The attributes `CaseSensitive` and `IgnoreWhiteSpaces` are not used during searches.

Example

```
<RB>
  <RegExpr>[&quot;' *value]</RegExpr>
</RB>
```

RB\Length Attribute

Value	Description
Type	Signed Numbers
Default	(empty)
Description	This attribute specifies a length value for an Offset/Length calculation, as described in section “Offset, Length”. The calculation is applied from the portion of an HTTP response where a rule hit begins through to the end of the response; it uses the offset of 0, and the given length. The end of the calculation's result marks the end of the rule hit.

Example

```
<RB>
  <Length>4</Length>
</RB>
```

RegExpr and RegExprTag Attribute

Value	Description
Type	Strings, Numbers
Default	(empty), 0
Description	These two attributes specify a regular expression that is used for searching rule hits. If the attribute `RegExprTag` is not omitted, it must specify the number of a tagged sub-expression within the given regular expression. Rule hits are searched for by applying the given regular expression to HTTP responses. Each substring of an HTTP response that matches the regular expression is a rule hit. If the attribute `RegExprTag` is specified, the rule hit is not the entire match, but the given tagged sub-expression.

Example

```
<RegExpr>name=&quot;\([0-9]+\)&quot;</RegExpr>
<RegExprTag>1</RegExprTag>
```

SearchIn Attribute

Value	Description
Type	Distinct Value Lists
Default	All

Allowed values are:

- All: This is an abbreviation for the complete list of other values.
- Header: Search HTTP response headers
- Body: Search HTTP response bodies
- HeaderName: Refers to the first parameter of the function `WebHeaderAdd`.
- HeaderValue: Refers to the second parameter of the function `WebHeaderAdd`.

This attribute specifies where to search for rule hits, either in response headers, response bodies, or both.

Example

```
<SearchIn>Body</SearchIn>
<SearchIn>Header</SearchIn>
```

CaseSensitive Attribute

Value	Description
Type	Boolean Values
Default	false
Description	This attribute specifies whether searches should be case-sensitive or caseinsensitive when searching the strings `LB\Str` and `RB\Str`. It also specifies if the option flag `WEB_FLAG_CASE_SENSITIVE` should be scripted when scripting the function `WebParseDataBoundEx`.

IgnoreWhiteSpaces Attribute

Shows the type and the default value of the attribute and provides a description.

Value	Description
Type	Boolean Values
Default	true
Description	This attribute specifies whether searches should ignore white spaces when searching the strings `LB\Str` and `RB\Str`.

Value	Description
	It also specifies whether to script the option flag `WEB_FLAG_IGNORE_WHITE_SPACE` when scripting the function `WebParseDataBoundEx`.

CondContentType Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute specifies a string that restricts rule hit searches to server responses that match the specified content type. The comparison is done using a prefixmatch.

Example

```
<CondContentType>text/</CondContentType>
```

This restricts the searching for rule hits to server responses with a content type such as "text/html" or "text/plain."

CondRegExpr Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute specifies a regular expression that is used to determine if a rule hit should be retained for future use. Each rule hit is checked to see if it matches the regular expression. Matching rule hits are retained, non-matching rule hits are dropped.

Example

```
<CondRegExpr>eCS@Store@[0-9]+-.*</CondRegExpr>
```

CondResultLen Attribute

Value	Description
Type	Numeric Ranges
Default	2-
Description	This attribute specifies a range that is used to determine if a rule hit should be retained for future use. If the number of bytes in the hit doesn't match the given range, the hit is dropped.

Conditions Attribute

Value	Description
Type	Structured Data
Description	This attribute specifies conditions that are applied to determine if a rule hit should be retained for future replacement. The conditions for each rule hit are evaluated within an environment that allows access to the HTTP request/ response that included the hit, in addition to other relevant data. If the conditions aren't determined to be true, the hit is dropped. See “Conditions” for more information regarding conditions. See the Section “ConditionEvaluation Environment” for more information regarding what may be subject to conditions.

Conversion\Dll Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute is the name of the conversion DLL used for modifying a parsed value. Make sure that the conversion DLL specified is added as a data file. The runtime searches for the DLL in the project directory and in the recording rules directory.

Conversion\Function Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute is the name of the conversion function exported by the conversion DLL.

Scripting Parsing Functions and Replacements

While the `Search` section specifies how to find rule hits in HTTP responses, the `ScriptGen` section specifies details regarding script generation. All XML paths of attributes described here are relative to: `HttpParsingRule\ScriptGen`.

The `ScriptGen` section allows users to specify conditions that restrict the usage of rule hits to script locations where replacement is appropriate, and to exclude locations where rule hits appear purely by coincidence. Additionally, some attributes can be used to instruct the Recorder as to which variable names to use and what comments are to be added to scripts to increase their readability.

VarName Attribute

Value	Description
Type	Strings
Default	(empty)
Description	This attribute specifies a variable name to script for the result of parsing functions. If omitted, the name of the rule is used as the variable name.

OnlyIfCompleteResult Attribute

Value	Description
Type	Boolean Value
Default	false
Description	This attribute specifies that a replacement should be scripted only if a complete string to be scripted can be replaced with a single variable. If false, replacements are scripted when a rule hit is a substring of the string to be scripted.

MaxLbLen Attribute

Value	Description
Type	Numbers
Default	Max unsigned
Description	When scripting a parsing function, the Recorder chooses a left boundary. This attribute can be used to specify a maximum length for the left boundary, in cases where there is danger that the left boundary may contain session information.

ReplaceIn Attribute

Value	Description
Type	Distinct Value Lists
Default	All
Description	This attribute specifies script locations that may contain hits to be replaced. Valid values are: • All: This is an abbreviation for the complete list of other values. • Url: Replace in URL parameters (various functions, for example `WebPageUrl`) • LinkName: Replace in link names (function `WebPageLink`). Link names that result from `WebPageParseUrl` are not replaced. • FormName: Replace in form names (function `WebPageSubmit`)

Value	Description
	• PostedData: Replace in binary posted data (various functions, for example `WebPagePost`) • FormFieldName: Form field name in the dclform section of the script • FormFieldValue: Form field value in the dclform section of the script • SetCookie: Parameter of the function `WebCookieSet` • HeaderName: Refers to the first parameter of the function `WebHeaderAdd`. • HeaderValue: Refers to the second parameter of the function `WebHeaderAdd`.

AlwaysNewFunc Attribute

Value	Description
Type	Boolean Values
Default	false
Description	This attribute specifies whether or not a rule hit, once parsed, can be reused or if the Recorder should script a new parsing function to parse the most recent occurrence of the string to be replaced.

CommentToVar Attribute

Value	Description
Type	Boolean Values
Default	true
Description	This attribute specifies whether or not a comment should be generated for the variable that contains the parsing result. If true, a comment that includes the value during recording is generated.

CommentToFunc Attribute

Value	Description
Type	Boolean Values
Default	false
Description	This attribute specifies whether or not a comment should be generated for the parsing function `WebParseDataBoundEx`. If true, a comment is generated during recording that includes the rule name that triggered the parsing function and the parsing result during recording.

GenBytesReadVar Attribute

Value	Description
Type	Boolean Values
Default	false
Description	This attribute specifies whether or not to generate a variable for the number of bytes parsed by a parsing function (see `WebParseDataBoundEx`, parameter `nBytesParsed`).

GenDebugPrint Attribute

Value	Description
Type	Boolean Values
Default	false
Description	This attribute specifies whether or not to generate a diagnostic Print function after the script function where a generated parsing function is in effect. If true, a Print function that prints the parsing result to the Silk Performer controller output window is scripted.

Conditions Attribute

Value	Description
Type	Structured Data
Description	This attribute specifies conditions that are applied to determine if a rule hit should be retained for future replacement. The conditions for each rule hit are evaluated within an environment that allows access to the HTTP request/ response that included the hit, in addition to other relevant data. If the conditions aren't determined to be true, the hit is dropped. See "Conditions" for more information regarding conditions. See the Section "ConditionEvaluation Environment" for more information regarding what may be subject to conditions.

Tokenizing of Rule Hits

The `Search` section used both in `HttpParsingRules` and `StringScriptingRules` contains a feature that allows to extract rule hits by tokenizing the search result. The idea is that each substring extracted (for example, using the various left/right boundary options) is not the rule hit itself, but can be "tokenized" to yield several rule hits. This tokenizing can be done in several ways and is specified by the xml tag `Tokenize`.

Valid values for the tag Tokenize are:

- `SiebelTokenHtmlSingleQuote`
- `SiebelTokenHtml`

- `siebelTokenApplet`

The tokenizing methods `SiebelTokenHtmlSingleQuote` and `SiebelTokenHtml` tokenize the search result into individual strings enclosed either in single or in double quotes. The tokenizing method `SiebelTokenApplet` tokenizes the search result assuming a series of length prefixed strings as used in applet responses in the Siebel 7 web application.

Example for SiebelTokenHtml

The search result `["TestName","TestSite","USD","02/21/2003","N","1-2T"]` will be tokenized and results in the following rule hits:

- TestName
- TestSite
- USB
- 02/21(2003
- N
- 1-2T

Example for SiebelTokenApplet

The search result `19*02/21/2003 08:20:176*SADMIN4*Note5*1-1P5` will be tokenized and results in the following rule hits:

- 02/21/2003 08:20:17
- SADMIN
- Note
- 1-1P5

The recorder will use these rule hits in the script by generating one of the tokenizing functions to extract the tokens at runtime. For a recording rule with tokenizing in the `Search` section:

```
<HttpParsingRule>
  <Name>Siebel Submit Data Array in HTML (from Javascript function call)</
Name>
  <Active>true</Active>
  <Search>
    <SearchIn>Body</SearchIn>
    <LB>
      <Str>SWESubmitForm</Str>
    </LB>
    <RB>
      <Str>'</Str>
    </RB>
    <Tokenize>SiebelTokenHtml</Tokenize>
    <CondResultLen>1-</CondResultLen>
  </Search>
  <ScriptGen>
    ...
  </ScriptGen>
</HttpParsingRule>
```

Script fragments from Siebel where tokenizing is used:

```
var
gsRowValArray_003 : string; // 0*19*02/21/2003 08:20:176*SADMIN4*Note5*1-1P5
// ...
WebParseDataBoundEx(gsRowValArray_003, sizeof(gsRowValArray_003),
"ValueArray`", WEB_OCCURENCE_LAST, "`",
WEB_FLAG_IGNORE_WHITE_SPACE, 1);
WebPageForm("http://lab72/sales_enu/start.swe", SALES_ENU_START_SWE026,
"Account Note Applet: InvokeMethod: NewRecord");
Print("Parsed \"RowValArray_003\", result: \"" + gsRowValArray_003 + "\"");
// Was "0*19*02/21/2003 08:20:176*SADMIN4*Note5*1-1P5" when recording
// ...
dclform
// ...
SALES_ENU_START_SWE027:
"SWEMethod" := "GetQuickPickInfo",
"SWEVI" := "",
"SWEView" := "Account Note View",
"SWEApplet" := "Account Note Applet",
"SWEField" := "s_2_2_24_0",
"SWER" := "0",
"SWEReqRowId" := "1",
"s_2_2_26_0" := "2/21/2003 08:20:17 AM",
"s_2_2_27_0" := SiebelTokenApplet(gsRowValArray_003, 2), // value: "SADMIN"
"s_2_2_24_0" := SiebelTokenApplet(gsRowValArray_003, 3), // value: "Note"
"s_2_2_25_0" := "",
"SWERPC" := "1",
"SWEC" := "11",
"SWEActiveApplet" := "Account Note Applet",
"SWEActiveView" := "Account Note View",
"SWECmd" := "InvokeMethod",
"SWERowId" := SiebelTokenApplet(gsRowValArray_003, 4),
// value: "1-1P5"
"SWERowIds" := "SWERowId0=" + SiebelTokenHtml(gsRowValArray_002, 18),
// value: "SWERowId0=1-2T"
"SWEP" := "",
"SWEJI" := "false",
"SWETS" := GetTimeStamp(); // value: "1045844419057"
```

Section ScriptGen

This section describes the `ScriptGen` section and the respective attributes.

MinRbLen LinLbLen

Value	Description
Type	Numbers
Default	1
Description	This option allows to specify a minimum length for the right / left boundary string which will be determined by the recorder and scripted as a parameter of the function call `WebParseDataBoundEx`.

Examples

```
<LB>
  <Str>name=&quot;</Str>
</LB>

<LB>
  <Str>BV_EngineID=</Str>
</LB>
```

LastOccurence

Value	Description
Type	Boolean Values
Default	false
Description	If this option is set to true, the recorder will script the constant WEB_OCCURENCE_LAST instead of the actual occurence of the left boundary, if the found occurence indeed was the last one during recording.

ExpectBinaryData

If this option is set to true, the recorder will generate a "bin-cast" for every use of the variable which is generated by this rule. This is necessary with the introduction of dynamic strings, if it is expected that the parsing result will contain binary data.

Instead of `"SomeText" + gsParsedValue + "some other text"`, the recorder will generate `"SomeText" + bin(gsParsedValue) + "some other text"`.

Default: false

Conversion\Bdl Attribute

This attribute is optional. It is the name of the BDL function wrapping the call of the conversion DLL function. If missing or left empty, the BDL wrapper function will have the same name as the `Conversion\Dll Attribute` function.

TCP/IP Protocol Rules

The Recorder uses TCP/IP protocol rules to detect proprietary TCP/IP protocols. If detected, the Recorder generates functions (depending on the protocol detected) that are better suited to handling dynamic server responses than is the `WebTcpipRecvExact` function, which is automatically scripted for unknown protocols.

Types of TCP/IP Protocol Rules

There are two types of TCP/IP protocol rules.

TcpRuleRecvProto

Use `TcpRuleRecvProto` rules to describe length-based protocols. Length-based protocols are protocols in which the number of bytes to be received from the server can be extracted from a fixed location in a protocol header. This rule type is specified by XML nodes with the name `TcpRuleRecvProto`.

TcpRuleRecvUntil

Use `TcpRuleRecvUntil` rules to describe protocols in which the end of a server response can be detected by a terminating sequence of bytes. This rule type is specified by XML nodes with the name `TcpRuleRecvUntil`.

Structure of TCP/IP Protocol Rules

Both types of TCP/IP protocol rules share the same basic structure. The Identify section contains attributes specific to the rule type. The Conditions section contains additional conditions that can be specified to avoid "detecting" protocols where they should not be detected (in cases where server responses coincidentally resemble protocols).

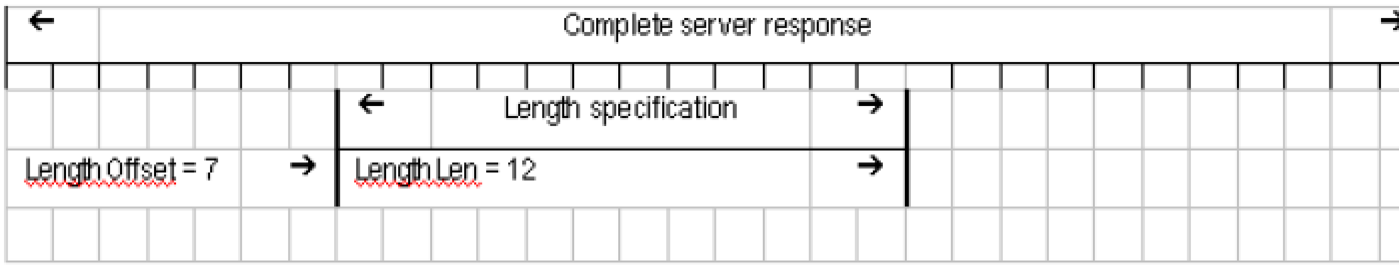
```
<TcpRuleRecvProto>
```

```
  <Name>Sample TCP RecvProto Rule</Name>
  <Active>true</Active>
  <Identify>
    …
  </Identify>
  <Conditions>
    …
  </Conditions>
</TcpRuleRecvProto>

<TcpRuleRecvUntil>
  <Name>Sample TCP RecvUntil Rule</Name>
  <Active>true</Active>
  <Identify>
    …
  </Identify>
  <Conditions>
    …
  </Conditions>
</TcpRuleRecvUntil>
```

TcpRuleRecvProto

The `TcpRuleRecvProto` rule type describes protocols with the following basic structure:



The number of bytes to be received can be extracted from the protocol header at offset `LengthOffset` using `LengthLen` number of bytes. This can be interpreted either in big endian or little endian representation. Additionally, the obtained value may be multiplied by a value (attribute `LengthFactor`), and/or a constant value may be added (attribute `LengthAdd`).

Identify\LengthOffset Attribute

Value	Description
Type	Numbers
Default	0

This attribute specifies the offset of the length specification within the protocol header. This corresponds to the parameter `nProtoStart` of the functions `WebTcpipRecvProto(Ex)`.

Identify\LengthLen Attribute

Value	Description
Type	Numbers
Default	4

This attribute specifies the number of bytes for the length specification within the protocol header. This corresponds to the parameter `nProtoLength` of the functions `WebTcpipRecvProto(Ex)`.

Identify\OptionFlags Attribute

Value	Description
Type	Distinct Value Lists
Default	(empty)

This attribute specifies how to interpret the length specification. Valid values correspond to the options available for the parameter `nOption` of the functions `WebTcpipRecvProto(Ex)`. The empty default value means: big endian.

Valid values are:

- LittleEndian: Option TCP_FLAG_LITTLE_ENDIAN
- ProtoIncluded: Option TCP_FLAG_INCLUDE_PROTO

Identify\LengthFactor Attribute

Value	Description
Type	Numbers
Default	1

This attribute specifies a factor. The protocol length is multiplied by this factor. This corresponds to the parameter `nMultiply` of the function `WebTcpipRecvProtoEx`.

Identify\LengthAdd Attribute

Value	Description
Type	Signed Numbers
Default	0

This property specifies a constant value. This value is added to the protocol length. This corresponds to the parameter `nSum` of the function `WebTcpipRecvProtoEx`.

Conditions

Additional conditions can be specified to exclude the "detection" of protocols in situations where server responses coincidentally resemble protocol specifications.

See “Conditions” for more information regarding conditions. See section “Condition Evaluation Environment” for more information regarding what may be subject to conditions.

GenVerify Attribute Of Conditions

Value	Description
Type	Boolean Values
Default	false

In the course of specifying basic conditions, the attribute `GenVerify` can be used to specify scripting for the parameters `sVerify`, `nVerifyStart` and `nVerifyLength` of the function `WebTcpipRecvProtoEx`.

When the conditions are evaluated, the first basic condition that results in an evaluation of true and specifies the attribute `GenVerify`, determines that the data to which this condition has been applied should be used to script the verification part of the function `WebTcpipRecvProtoEx`.

Guided TcpRuleRecvProto Example

This example illustrates how to write a recording rule for the Siebel 6 Thin Client.

This client is an ActiveX control that runs in a Web browser. A recorded script consists of one `WebPageUrl()` call followed by TCP/IP traffic from the ActiveX control. Without recording rules, the server responses are scripted by `WebTcpipRecvExact()` calls.

Portion of a Recorded Siebel 6 TCP/IP Script

```
WebTcpipSendBin(hWeb0,
  "\h0000003000000000000000000000001" // ···0··········· 00000000
  "\h0000000C0000001C0000019100000000" // ················ 00000010
  "\h0000002000000258000000C00000000" // ··· ···X········ 00000020
  "\h00000000", 52); // ···· 00000030
WebTcpipRecvExact(hWeb0, NULL, 40);
```

Server responses can be analyzed with the help of TrueLog Explorer.

Each response contains a protocol header consisting of 4 bytes that specify the number of bytes following the protocol header. This length specification is in big endian notation (most significant byte first).

With these findings a recording rule can be written, as shown below.

Recording Rule for Siebel 6 Thin Client TCP/IP Traffic

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>
  <TcpRuleRecvProto>
    <Name>Siebel TCP Protocol</Name>
    <Identify>
      <LengthOffset>0</LengthOffset>
      <LengthLen>4</LengthLen>
    </Identify>
  </TcpRuleRecvProto>
</RecordingRuleSet>
```

This rule specifies that the protocol header contains the length of the data block at offset 0 using 4 bytes. Using this rule, the Recorder generates scripts that use the function `WebTcpipRecvProto()` for the server responses, as shown below.

Portion of a Recorded Script using Recording Rules

```
WebTcpipSendBin(hWeb0,
  "\h0000003000000000000000000000001" // ···0··········· 00000000
  "\h0000000C0000001C0000019100000000" // ················ 00000010
  "\h0000001F000002580000000C00000000" // ·······X········ 00000020
  "\h00000000", 52); // ···· 00000030
WebTcpipRecvProto(hWeb0, 0, 4);
```

Scripts recorded with this rule replay correctly even when the number of bytes to be received differs from the number of bytes received during recording.

TcpRuleRecvUntil

The `TcpRuleRecvUntil` rule type specifies protocols whereby the end of a server response can be detected by searching for a terminating byte sequence.

Identify\TermData Attribute

Value	Description
Type	Binary Data
Default	(empty)

This attribute specifies the terminating byte sequence of the protocol.

Identify\IgnoreWhiteSpaces Attribute

Value	Description
Type	Boolean Values
Default	false

This attribute specifies whether or not the Recorder should ignore white spaces while searching for terminating data. Scripted terminating data (parameter `sPattern` of the function `WebTcpipRecvUntil`) exactly reflects what was seen during recording and therefore, with respect to white spaces, may differ from what is specified in the property `Identify\TermData`.

Conditions

Additional conditions can be specified to exclude the "detection" of protocols in situations where server responses coincidentally resemble protocol specifications.

See “Conditions” for more information regarding conditions. See section “Condition Evaluation Environment” for more information regarding what may be subject to conditions.

Guided TcpRuleRecvUntil Example

This example shows how to improve recorded scripts for telnet sessions.

Recording a telnet session without recording rules results in scripts that have lots of `WebTcpipRecvExact()` function calls, as shown below.

Portion of a recorded Telnet script without recording rules

```
WebTcpipSend(hWeb0, "l");
WebTcpipRecvExact(hWeb0, NULL, 1);
WebTcpipSend(hWeb0, "s\r\n");
WebTcpipRecvExact(hWeb0, NULL, 1);
WebTcpipRecvExact(hWeb0, NULL, 2);
WebTcpipRecvExact(hWeb0, NULL, 1220);
```

TrueLog Explorer is used to analyze server responses. Each server response ends with a command prompt. The command prompt ends with the three-character sequence blank, hash, blank, which seems a reasonable choice for the terminating data of a server response.

A recording rule can be written for this kind of server response. The rule instructs the Recorder to watch for server responses that end with the special terminating sequence, and to script the function `WebTcpipRecvUntil()` instead of `WebTcpipRecvExact()` for those server responses.

TCP/IP recording rule for Telnet

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>
  <TcpRuleRecvUntil>
    <Name>Telnet Command Prompt</Name>
    <Active>true</Active>
    <Identify>
      <TermData> # </TermData>
      <IgnoreWhiteSpaces>false</IgnoreWhiteSpaces>
    </Identify>
    <Conditions>
      <NoBlockSplit>true</NoBlockSplit>
    </Conditions>
  </TcpRuleRecvUntil>
</RecordingRuleSet>
```

The terminating data is specified in the attribute `Identify\TermData`. Since the terminating data contains significant white spaces, not to ignore white spaces must be specified in the attribute `Identify\IgnoreWhiteSpaces`.

The rule should not be applied if a server response coincidentally contains the same terminating sequence elsewhere in the response. Therefore the condition `NoBlockSplit` should be specified. This means that the end of the terminating data must be aligned with the end of a TCP/IP packet received from the telnet server; otherwise the rule won't be applied.

Portion of a recorded Telnet script using a recording rule

```
WebTcpipSend(hWeb0, "l");
WebTcpipRecvExact(hWeb0, NULL, 1);
WebTcpipSend(hWeb0, "s");
WebTcpipRecvExact(hWeb0, NULL, 1);
WebTcpipSend(hWeb0, "\r\n");
WebTcpipRecvUntil(hWeb0, NULL, 0, NULL, " # ");
```

This example shows that the rule works. This script is better equipped to handle varying server responses during replay.

Note that the terminating sequence may be different for other Telnet servers and may depend on operating system and user settings. Therefore the rule shown here is not a general rule that works for all Telnet servers. It can however easily be adapted by changing the terminating character sequence. With the help of TrueLog Explorer, it's easy to determine suitable terminating byte sequences for other Telnet servers.

StringScriptingRule

The purpose of the `StringScriptingRule` is to hook into the process of scripting strings.

Whenever the Web recorder generates a string into the script (no matter where, this can be any parameter of an API function or a form field name or value), rules of this type are evaluated and may result in special actions, so that the string is not generated "as is", but processed in some way.

Structure

The basic structure of a `StringScriptingRule` is very similar to the structure of a `HttpParsingRule`. It also consists of two sections named `Search` and `ScriptGen` (see example below).

```
<StringScriptingRule>
  <Name>Replace TimeStamp</Name>
  <Active>true</Active>
  <Search>
    <SearchIn>FormFieldValue</SearchIn>
    <LB>
      <Offset>0</Offset>
    </LB>
    <RB>
      <Length>0</Length>
    </RB>
  </Search>
  <ScriptGen>
```

```
    <Action>CheckTimeStamp</Action>
  </ScriptGen>
</StringScriptingRule>
```

Section Search

While the details of the Search section of a `HttpParsingRule` are applied to any HTTP response (with the goal to extract rule hits which are candidates for future replacements), the `Search` section of a `StringScriptingRule` is applied to the string being scripted, with the goal to extract substrings that should be treated in a special way. All techniques how substrings can be extracted (see “Section Search - Finding Rule Hits”) can also be used in the Search section of a `StringScriptingRule`.

The only difference is the meaning of the attribute `SearchIn`:

For a `HttpParsingRule`, the attribute `SearchIn` specifies where to look for rule hits. Allowed values are: Header, Body, All.

For a `StringScriptingRule`, the attribute `SearchIn` specifies a list of script locations where this rule should be applied. This can be a list of script locations for the condition type `Scripting`, as described in “Scripting Condition”.

Additionally, there are two new script locations `HeaderName` and `HeaderValue`, which identify the two parameters of the function `WebHeaderAdd`. See “Additional ApplyTo values” for detailed information.

There is one additional possibility how to search for substrings (which is also applicable in the Search section of a `HttpParsingRule`):

The attribute `Special` allows to specify a special search algorithm. The only value currently supported is `SiebelParam`. This will search the string for any substring that looks like a length-prefixed value as it is used in the Siebel 7 Web application.

```
<StringScriptingRule>
  <Name>SiebelParam Function Call</Name>
  <Active>true</Active>
  <Search>
    <SearchIn>FormFieldValue</SearchIn>
    <Special>SiebelParam</Special>
    <Conditions>
    </Conditions>
  </Search>
  <ScriptGen>
    <Action>CheckSiebelParam</Action>
  </ScriptGen>
</StringScriptingRule>
```

Section ScriptGen

This section describes the `ScriptGen` section and the respective attributes.

Attribute Action

This attribute specifies which action should be taken for the substring identified by the `Search` section.

The following values are allowed:

Value	**Description**
CreateVariable	This will create a variable initialized to the value of the string, and the variable will be substituted in the script instead of the original value.

Value	Description
CheckTimeStamp	This will create the function call `GetTimeStamp()` instead of the original string, if the original string indeed looks like a plausible timestamp value.
CheckSiebelParam	Checks if the original string is structured like a length prefixed value like used in the Siebel 7 Web application. If so, it will replace the original value with the function call `Siebel7Param(...)`. Example: `"5*Value" will become: Siebel7Param("Value")`
DecodeSiebelJavaScriptString	Checks if the original string contains special characters that are usually escaped with a backslash within JavaScript code, modifies the string in a way it would be if it was embedded in JavaScript code, and creates the function `SiebelDecodeJsString(...)` with the encoded string as parameter. The function `SiebelDecodeJsString` will reverse this encoding during script replay, so that the same network traffic is generated. The purpose of this is that the modified parameter may now be parseable by `HttpParsingRules`, which might not be possible without this substitution.
CheckSiebelDateTime	Checks if the original string looks like a date/time combination in the format which is sent by the Siebel Web client, and transforms it to an equivalent date/ time combination in the format which appears in server responses to the Siebel Web client. If this is the case, the wrapper function `SiebelDateTime` is recorded which undoes this transformation during script replay. The purpose of this is that date/time combinations can then be parsed by the other parsing rules, because they appear in the script in the same format as they appear in server responses.
CheckSiebelDate	The same as `CheckSiebelDateTime`, but for dates only. Records the wrapper function `SiebelDate`.
CheckSiebelTime	The same as `CheckSiebelDateTime`, but for times only. Records the wrapper function `SiebelTime`.
CheckSiebelPhone	The same as `CheckSiebelDateTime`, but for phone numbers. Records the wrapper function `SiebelPhone`.

For examples, see the recording rules of the Siebel 7 Web SilkEssential.

Example (create variable for the quantity in an online shop): This example assumes that the quantity is sent in a form field named `qty`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<RecordingRuleSet>
  <StringScriptingRule>
    <Name>Parameterize item quantity</Name>
    <Active>true</Active>
    <Search>
      <SearchIn>FormFieldValue </SearchIn>
      <LB>
        <Offset>0</Offset>
      </LB>
      <RB>
        <Length>0</Length>
      </RB>
      <Conditions>
        <CompareData>
          <ApplyTo>FormFieldName</ApplyTo>
          <Length>0</Length>
          <Data>qty</Data>
        </CompareData>
      </Conditions>
    </Search>
    <ScriptGen>
      <Action>CreateVariable</Action>
      <VarName>Quantity</VarName>
    </ScriptGen>
  </StringScriptingRule>
</RecordingRuleSet>
```

Attributes VarName VarNamePrefix IsExternalVar

These attributes are only applicable if the Action attribute is `CreateVariable`. This allows to specifiy either a variable name (attribute `VarName`) or a prefix for the variable name (attribute `VarNamePrefix`). In the latter case, the actual variable name will be built from the given prefix and the actual string value during recording.

The attribute `IsExternalVar` (boolean, Default: false) can be used to suppress the scripting of a declaration for the variable. This is useful if it is known that this variable is already declared in some bdh-file within a SilkEssential.

HttpScriptingRule

The rule type `HttpScriptingRule` allows to hook into script generation decisions the recorder has to make. It allows to override the default heuristics the recorder employed prior to this rule type and still employs in the absence of such rules.

Structure

The basic structure of a `HttpScriptingRule` is quite simple. The only tag allowed, additional to the common tags `Name` and `Active`, is the tag `Action`. The `Action` tag specifies which decision is to be hooked.

The actual decision is implemented with conditions. The result of evaluating the conditions is the return value of such a rule. Conditions have access to the HTTP request / response which is being scripted.

```
<HttpScriptingRule>
  <Name>Suppress some cookie</Name>
  <Active>true</Active>
  <Action>SuppressCookie</Action>
  <Conditions>
    ...
  </Conditions>
</HttpScriptingRule>
```

HttpScriptingRule Actions

ForceTcp

This recording rule ensures that a script is generated using TCP/IP low-level functions. No protocol detection is performed. The recording rule only applies to scripts that are generated out of capture files.

```
<RecordingRuleSet>
  <HttpScriptingRule>
    <Name>Force TCP/IP Scripting</Name>
    <Active>true</Active>
    <Action>ForceTcp</Action>
  </HttpScriptingRule>
</RecordingRuleSet>
```

NoHtml ForceHtml

The recorder will evaluate `HttpScriptingRules` with Action `NoHtml` or `ForceHtml` whenever it needs a decision if a HTTP response body is HTML or not. In the absence of such rules or if no such rules return true, the recorder inspects the content-type header of the HTTP response.

This rule type is useful to suppress or force the scripting of a page-level function in cases where the default recording result is not satisfactory.

Often a "404 Not Found" response comes back with an HTML error description, which will cause the scripting of a `WebPageUrl` or `WebPageLink`, even when a `WebPageAddUrl` would be more appropriate.

```

<HttpScriptingRule>
  <Name>No HTML for zip files</Name>
  <Active>true</Active>
  <Action>NoHtml</Action>
  <Conditions>
    <CompareData>
      <ApplyTo>Http.Initial.Request.Url.Ext</ApplyTo>
      <Data>zip</Data>
    </CompareData>
  </Conditions>
</HttpScriptingRule>
```

NoFuzzyFormDetection ForceFuzzyFormDetection

These actions of a `HttpScriptingRule` can be used to override the setting **Fuzzy form detection** from the profile settings for individual HTTP requests.

This rule allows fuzzy form detection (provided this is enabled in the profile settings) only for forms which contain a form field named `sid`.

```

<HttpScriptingRule>
  <Name>No fuzzy form detection except for forms with field
sid</Name>
  <Active>true</Active>
  <Action>NoFuzzyFormDetection</Action>
  <Conditions>
    <Not>
      <Exists>
        <ApplyTo>Form.Field(Name:sid)</ApplyTo>
      </Exists>
    </Not>
  </Conditions>
</HttpScriptingRule>
```

NoDynLinkParsing ForceDynLinkParsing

These Actions of a `HttpScriptingRule` can be used to override the setting "Dynamic link parsing" from the profile settings for individual HTTP requests.

This example forces dynamic link parsing (even if it turned off in the profile settings) for HTTP requests where the query string contains the string `sid=`.

```
<HttpScriptingRule>
  <Name>Force dynamic link parsing for some query strings</Name>
  <Active>true</Active>
  <Action>ForceDynLinkParsing</Action>
  <Conditions>
    <FindData>
      <ApplyTo>Http.Initial.Request.Url.QueryData</ ApplyTo>
      <Data>sid=</Data>
    </FindData>
  </Conditions>
</HttpScriptingRule>
```

DefinePageName

This action allows defining an alternate page name for a HTML page. The recorder generates various strings (page timer name, name of stored context variable, ...) based on the name of a HTML page. By default the recorder uses the title of an HTML page for the page name, or "Unnamed page" if no title exists.

Such a rule must have conditions which:

- return true
- save a non-empty string to the variable `PageName` (by means of the tag `SaveAs` of a condition)

The conditions have access to the default page name the recorder would use through the variable `DefaultPageName`.

The following example checks if the default page name does not exist (the recorder would use "Unnamed page" then), and defines the page name to be the URL of the HTTP document instead, if it is at least 3 characters long.

```
<HttpScriptingRule>
  <Name>Define Page Name</Name>
  <Active>true</Active>
  <Action>DefinePageName</Action>
  <Conditions>
    <Not>
      <Exists>
        <ApplyTo>DefaultPageName</ApplyTo>
      </Exists>
    </Not>
    <CheckRange>
      <ApplyTo>Http.Initial.Request.Url</ApplyTo>
      <Range>3-</Range>
      <SaveAs>PageName</SaveAs>
      <SaveMode>Replace</SaveMode>
    </CheckRange>
  </Conditions>
</HttpScriptingRule>
```

SuppressCookie CommentCookie

In some cases the function `WebCookieSet` may be recorded where this is not useful or even incorrect.

This may happen when there is a tight succession of client side "Cookie" and server side "Set-Cookie" headers in embedded objects of a page, with tight timing. In such a case it is possible that a browser does not send a cookie or sends a cookie with an older value, although the browser already received a "Set-Cookie" header which should cause it to send a different cookie value.

In such cases, these actions can be used to script `WebCookieSet` function calls commented or suppress the recording completely.

This rule suppresses scripting of any `WebCookieSet` function call if the cookie name is PS_TOKENEXPIRE (from the Peoplesoft SilkEssential).

```
<HttpScriptingRule>
  <Name>Suppress Cookie PS_TOKENEXPIRE</Name>
  <Active>true</Active>
  <Action>SuppressCookie</Action>
  <Conditions>
    <CompareData>
      <ApplyTo>Cookie</ApplyTo>
      <Data>PS_TOKENEXPIRE</Data>
    </CompareData>
  </Conditions>
</HttpScriptingRule>
```

ScriptCookieDomainAsRequestHost

Usually, when recording a client-side cookie, Silk Performer uses a generic form for the `domain` attribute. The domain is cropped by the first dot, for example: `http://demo.borland.com/testsite` will be cropped to `.borland.com`. Usually, this is the desired behaviour, because such a cookie is valid for a variety of domains, including `demo.borland.com` and `example.borland.com` regarding the above example.

The recording rule `ScriptCookieDomainAsRequestHost` defines the domain for a cookie more precisely. The domain attribute matches exactly the request host.

The recording rule can be applied to certain cookies, by adding the name of the cookie to the `<data>` tag.

```
<HttpScriptingRule>
  <Name>MyCookieRecordingRule</Name>
  <Active>true</Active>
  <Action>ScriptCookieDomainAsRequestHost</Action>
  <Conditions>
    <CompareData>
      <ApplyTo>Cookie</ApplyTo>
      <Data>MyClientsideTestCookie</Data>
    </CompareData>
  </Conditions>
</HttpScriptingRule>
```

SuppressFrameName SuppressLinkName SuppressFormName SuppressCustomUrlName

Functions like `WebPageLink`, `WebPageSubmit` and `WebPageSetActionUrl` use a name (of a link, form or parsed URL resp.), a frame name, and an ordinal number to reference a link, form, or parsed URL.

In cases where for example a link or form name changes dynamically, it is common practice to do the so-called "name to number customization", which means one does not specify the name (of the link or form), but instead specifies the overall ordinal number within the page or within the frame.

These actions can be used to let this customization be done already by the recorder.

Assume a shop application which uses form names which are built by adding the numeric session ID to the string `cart_`.

This rule will not record the form name within the `WebPageSubmit` calls, but will record NULL instead, with an ordinal number which references the form in the entire page.

```
<HttpScriptingRule>
  <Name>Suppress XML to form conversion</Name>
  <Active>true</Active>
  <Action>SuppressFormName</Action>
  <Conditions>
    <RegExpr>
      <ApplyTo>FormName</ApplyTo>
      <Data>cart_[0-9]*</Data>
      <ExpectMatch>Complete</ExpectMatch>
    </RegExpr>
  </Conditions>
</HttpScriptingRule>
```

MapFunctionName

This action can be used to define a function name mapping, so that the recorder scripts a wrapper function (which needs to be present, for example in a BDH file, for the script to be compileable) instead of the original function. Such a wrapper function must have the same parameter list as the original function, because this rule does not modify the function parameters in any way.

The recorder scripts the wrapper function instead of the original function, if the condition returns true and in the course of the evaluation of the condition something was saved to the variable `FunctionName`.

If the conditions also save a non-empty string to the variable `BdhFileName`, the recorder will also generate a use statement to include the given BDH file into the script.

This rule replaces each `WebPageLink` function call with a `MyWebPageLink` function call.

```
<HttpScriptingRule>
  <Name>Replace WebPageLink with my wrapper function</Name>
  <Active>true</Active>
  <Action>MapFunctionName</Action>
  <Conditions>
    <CompareData>
      <ApplyTo>DefaultFunctionName</ApplyTo>
      <Data>WebPageLink</Data>
    </CompareData>
    <Exists>
      <ApplyTo>Literal:MyWebPageLink</ApplyTo>
      <SaveAs>FunctionName</SaveAs>
    </Exists>
    <Exists>
      <ApplyTo>Literal:MyFunctions.bdh</ApplyTo>
      <SaveAs>BdhFileName</SaveAs>
    </Exists>
  </Conditions>
</HttpScriptingRule>
```

AddInitFunction

This action can be used to add function calls to the TInit or TMain transaction. This is the only action, which allows additional tags, besides the `Conditions` tag. While conditions are allowed, they are not useful because they can not reference any data.

There can be any number of functions specified, each one with its own `Function` tag. Within the `Function` tag, there must be the tag `FunctionName` and optionally the tag `BdhFileName`.

There can be any number of `Param` tags to specify parameters. Each parameter must have a `Value` tag and a `Type` tag. The `Type` tag can be either `String` or `Const`. `String` will put the parameter in quotes, `Const` won't.

```
<HttpScriptingRule>
  <Name>Automatically insert my useful function</Name>
  <Active>true</Active>
  <Action>AddInitFunction</Action>
  <Function>
    <FunctionName>InitMyLogLibrary</FunctionName>
    <BdhFileName>Log.bdh</BdhFileName>
    <Param>
      <Value>C:\Logs\xx.log</Value>
      <Type>String</Type>
    </Param>
    <Param>
      <Value>true</Value>
      <Type>Const</Type>
    </Param>
  </Function>
</HttpScriptingRule>
```

This will generate the following script fragments:

```
use "Log.bdh"

transaction TInit
begin
  // ...
  InitMyLogLibrary("C:\\Logs\\xx.log", true);
end TInit;
```

Silk Performer allows the tag `Location` with the following values:

- `TInit`
- `TMainBegin`
- `TMainEnd`

NoAddUrl

This action can be used to suppress the generation of a `WebPageAddUrl` function for particular HTTP requests. The recorder will then choose the next best way to record this request, usually a `WebUrl` function will be scripted instead, but it might also be another function (for example `WebPageLink`, `WebPageUrl`, ...) depending on circumstances.

This can be useful, because parsing rules never parse data from embedded objects of a web page. If a HTTP request is scripted by a `WebUrl` function call instead, it is possible to apply HTTP parsing rules to this request.

SuppressRecording

This action can be used to suppress the recording of individual HTTP requests. It is used by the Silk Performer Oracle Forms support to suppress the recording of HTTP requests which come from an Oracle Forms applet which is recorded at a different API level.

ForceContextless

This action can be used to suppress the recording of HTTP requests with a context-full function like `WebPageLink` or `WebPageSubmit`. Instead, the recorder will record an appropriate context-less function instead.

DontUseHtmlVal

This action can be used to suppress the recording of the <USE_HTML_VAL> tag for individual form fields, even if the recorder would do so otherwise.

This can be useful if the form field value is required in the script, for example because another recording rule should be applied to the value.

In the `PeopleSoft` project type, the user name in the login form is forced to be in the script with this rule (even if the user name input field is pre-populated), and then another rule of type `StringScriptingRule` with the action `CreateVariable` creates a variable for it. In this way the script is better prepared for randomization of the login data.

NoQueryToForm ForceQueryToForm

These actions can be used to override the profile setting **Convert URL query strings to forms** on a per HTTP request basis.

ProxyEngineRule

The rule type `ProxyEngineRule` allows to control some aspects of the ProxyEngine's operation.

Structure

The basic structure of a `ProxyEngineRule` is quite simple. The only tag allowed, additional to the common tags `Name` and `Active`, is the tag `Action`. The `Action` tag specifies what to do. The actual decision is implemented with conditions. The result of evaluating the conditions is the return value of such a rule.

```
<ProxyEngineRule>
  <Name>Switch on GUI recording for Jacada</Name>
  <Active>true</Active>
  <Action>AddAppletParam</Action>
  <Conditions>
    <FindData>
      <ApplyTo>Attribute.ARCHIVE</ApplyTo>
      <Data>clbase.jar</Data>
    </FindData>
    <Exists>
      <ApplyTo>Literal:GUIMode</ApplyTo>
      <SaveAs>Param.Name</SaveAs>
      <SaveMode>Replace</SaveMode>
    </Exists>
    <Exists>
      <ApplyTo>Literal:GUIRecorder</ApplyTo>
      <SaveAs>Param.Value</SaveAs>
      <SaveMode>Replace</SaveMode>
    </Exists>
  </Conditions>
</ProxyEngineRule>
```

ProxyEngineRule Actions

AddAppletParam

This `Action` allows modifying applets contained in HTML before the HTML is forwarded to the client. The modification will not be visible in the log files. If the conditions evaluate to true, and the conditions save values to `Param.Name` and `Param.Value`, this parameter will be added to the applet tag.

Any `ProxyEngineRule` with Action `AddAppletParam` will be evaluated once per applet. The conditions have access to the attributes of the applet using the following syntax: `Attribute.attr`. `attr` may be any attribute of the applet tag.

RemoveAppletParam ChangeAppletParam

This action allows modifying applets contained in HTML before the HTML is forwarded to the client. The modification will not be visible in the log files. If the conditions evaluate to true, the parameter will be removed from the applet (`RemoveAppletParam`), or its value will be changed to whatever was saved by the conditions to `Param.Value` (`ChangeAppletParam`).

Any `ProxyEngineRule` with one of these actions will be evaluated once per applet parameter. The conditions can access the name and value of the applet parameter by `Param.Name` and `Param.Value`.

```
<ProxyEngineRule>
  <Name>Remove Cabbase applet param</Name>
  <Active>true</Active>
  <Action>RemoveAppletParam</Action>
  <Conditions>
    <Or>
      <CompareData>
        <ApplyTo>Param.Name</ApplyTo>
        <Data>Cabbase</Data>
        <Length>0</Length>
      </CompareData>
      <CompareData>
        <ApplyTo>Param.Value</ApplyTo>
        <Data>.cab</Data>
        <Offset>-4</Offset>
      </CompareData>
    </Or>
  </Conditions>
</ProxyEngineRule>
```

DetectProtoFtp DetectProtoSmtp

In the course of protocol detection, it is sometimes hard do distinguish between FTP and SMTP, since these protocols start with very similar traffic. The actions `DetectProtoFtp` and `DetectProtoSmtp` can be used to force detecting FTP or SMTP in cases where the recorder would misdetect the protocol otherwise.

The conditions can assess the following pieces of information in the `ApplyTo` tag:

- `WelcomeMsg`: The welcome message of the server
- `NoopResponse`: The server response to a NOOP command which is sent by the ProxyEngine

- `TargetPort`: The target port

DontModifyRequestHeader DontModifyResponseHeader

The recorder routinely modifies some request/response headers of HTTP traffic to ensure best recording results with common browsers. However, some uncommon user agents may misbehave when recorded.

An `Action` enables you to suppress the modification of request/response HTTP headers for individual requests, and thus allow for such recording problems.

Conditions can reference the header name and value, which can be modified in the `ApplyTo` tag using the values `HeaderName` and `HeaderValue`.

This example is from the Flex/AMF3 project type. This rule suppresses the modification of the `Pragma` and `Cache-Control` response header, if there is no `Accept-Language` request header.

```
<ProxyEngineRule>
  <Name>Suppress modification of some server response headers
for HTTP requests coming from Shockwave/Flash</Name>
  <Active>true</Active>
  <Action>DontModifyResponseHeader</Action>
  <Conditions>
    <Not>
      <Exists>
        <ApplyTo>Http.Initial.Request.Header.Accept-Language</
ApplyTo>
      </Exists>
    </Not>
    <Or>
      <CompareData>
        <ApplyTo>HeaderName</ApplyTo>
        <Data>Pragma</Data>
        <Length>0</Length>
      </CompareData>
      <CompareData>
        <ApplyTo>HeaderName</ApplyTo>
        <Data>Cache-Control</Data>
        <Length>0</Length>
      </CompareData>
    </Or>
  </Conditions>
</ProxyEngineRule>
```

DontDetectProtoHttp

Sometimes it is necessary to record a TCP connection on TCP/IP level, although the automatic protocol detection would detect HTTP. The Actions `DontDetectProtoHttp` can be used to suppress detecting HTTP in such cases.

The conditions can assess the following pieces of information in the `ApplyTo` tag:

- `PeekData`: The first chunk of data sent by the client.

BypassBlockingHttp

Sometimes the recorder must be stopped from analyzing network traffic so as to avoid blocking network communication. Use the action `BypassBlockingHttp` to suppress recording in such cases. When this action is used, data on the connection is still forwarded, but it is not recorded until the connection is closed.

Conditions assess the following pieces of information in the `ApplyTo` tag:

- `PeekData`: The relative request URL contains a specific value.

Conditions

Conditions are an important aspect of `HttpParsingRule`, `TcpRuleRecvProto` and `TcpRuleRecvUntil` rule types.

They are a powerful means of specifying exactly when rules should be applied.

Introduction

Compound Conditions

Complex conditions can be created using Boolean operators And, Or and Not.

```
<And>
  <Or>
    <SomeBasicCondition> … </SomeBasicCondition>
    <SomeBasicCondition> … </SomeBasicCondition>
    <Not>
      <SomeBasicCondition> … </SomeBasicCondition>
    </Not>
  </Or>
  <Not>
    <And>
      <SomeBasicCondition> … </SomeBasicCondition>
      <SomeBasicCondition> … </SomeBasicCondition>
    </And>
  </Not>
</And>
```

Extended Boolean Values

The result of a condition, once evaluated, is an extended Boolean value. Extended Boolean values have the values true, unknown or false. Think of the value unknown as: "Cannot be evaluated now, but may be evaluated when more data becomes available".

This is important for `TcpRuleRecvProto` and `TcpRuleRecvUntil` type rules. If conditions in a TCP rule result in a value of unknown, the Recorder defers scripting and reevaluates conditions when more data arrives from the server.

Basic Conditions

There are a number of basic condition types that execute checks and can be combined (using the Boolean conditions And, Or and Not) to build complex compound conditions.

Basic condition types include:

- CheckRange: Checks to see if a numeric value lies within a given range.
- ResultLen: A special form of the condition CheckRange.
- CompareData: Compares data.
- FindData: Searches data.
- Verify: A special form of the condition CompareData.
- RegExpr: Applies a regular expression.
- NoBlockSplit: Checks block boundaries.
- Scripting: Checks for the type of string being scripted.

Condition evaluation environment

A condition is evaluated within an environment. Through the environment, the condition has access to a number of strings to which the condition can be applied. Environment configuration differs with each rule type. See section “Condition Evaluation Environment” for details.

Conditions operate on data

Most conditions (except the `Scripting` condition) apply specific checks on specific blocks of data. There are flexible means of specifying what data is to be checked. See section “Specifying Data for Conditions” for more information.

Specifying Data for Conditions

Most conditions (except the `Scripting` condition) operate on specific blocks of data.

Conditions have sets of attributes that specify what data is to be used and what should be done if required data is not available.

These attributes are:

- `ApplyTo`
- `Offset`
- `Length`
- `IfNull`
- `UseDataAvail`

The basic idea is:

1. `ApplyTo` specifies a block of data (for example a request URL, a response body, the string to be scripted, and so on).
2. `Offset` and `Length` are optionally used to target data subsets.

ApplyTo

Value	Description
Type	Strings
Default	Self
Description	`ApplyTo` specifies the data that the condition operates on. `ApplyTo` is resolved within the current environment. The valid values for `ApplyTo` are therefore dependent on the environment configuration, which is different for each rule type.

Literal

With the Literal value in the `ApplyTo` tag it is possible to specify a literal as data for conditions. This is useful in conjunction with the `Exists` condition and the `SaveAs` attribute.

```
<Conditions>
  <Exists>
    <ApplyTo>Literal: - </ApplyTo>
    <SaveAs>PageName</SaveAs>
    <SaveMode>AppendSeparator</SaveMode>
  </Exists>
  <Exists>
    <ApplyTo>Form.Submit.Field(Name:SWECmd).Value</ApplyTo>
    <SaveAs>PageName</SaveAs>
    <SaveMode>Append</SaveMode>
  </Exists>
</Conditions>
```

Additional ApplyTo values

Additional `ApplyTo` values in the `ScriptGen` section of a `HttpParsingRule`.

In addition to the possible values `LinkName`, `FormName`, `FormFieldName`, `FormFieldValue` and `TargetFrame` in the `ScriptGen` section of a `HttpParsingRule`, there are four new possibilities:

CustomUrlName	The current name of a custom URL, if used in a function `WebPageLink`, `WebPageSetActionUrl` or `WebPageQueryParsedUrl`.
PostedData	The value of (possibly binary) posted data that appears as a parameter to `WebPagePostBin` and similar functions. The name of a HTTP header, when the function `WebHeaderAdd` is recorded (1st parameter of this function).
HeaderName	The name of a HTTP header, when the function `WebHeaderAdd` is recorded (1st parameter of this function).
HeaderValue	The value of a HTTP header, when the function `WebHeaderAdd` is recorded (2nd parameter of this function).

Access to forms

Both rules of type `HttpParsingRule` (`ScriptGen` section), `HttpScriptingRule` and `StringScriptingRule` (`Search` section) also have convenient access functions for form data according to the following syntax: `"Form" [ ".Query" | ".Body" | ".Submit" ]`

This allows specifying the form in the query string ("Query") or in the request body ("Body") of the current HTTP request.

The value `Submit`, which is the default, is automatically equivalent to Query if the HTTP request uses the method GET, and to Body if the HTTP request uses the method POST.

Once a form is specified according to the syntax above, it is possible to extract details about this form:

- `ActionUrl`: Specifies the action URL of a form. It is possible to get more details of the action URL.
- `Encoding`: Specifies the encoding to be scripted (for example `ENCODE_BLANKS`)

```
<ApplyTo>Form.ActionUrl</ApplyTo>
<ApplyTo>Form.ActionUrl.Host</ApplyTo>
<ApplyTo>Form.ActionUrl.Coords</ApplyTo>
<ApplyTo>Form.Query.Encoding</ApplyTo>
<ApplyTo>Form.Body.Encoding</ApplyTo>
<ApplyTo>Form.Submit.Encoding</ApplyTo>
```

Moreover, it is possible to specify the individual form fields according to the following syntax: `".Field(" ( "Name" | "Value" | "Index" ) ":" Selector ")" "." ( "Name" | "Value" | "Encoding" | "Usage" )`

So it is possible to reference a form field by name, value, or index (zero-based), and extract the name, value, encoding, or usage of such a form field. Encoding means one of the following: "ENCODE_FORM". Usage means one of the following: "SUPPRESS", "USE_HTML_VAL", "USE_SCRIPT_VAL".

```
<ApplyTo>Form.Submit.Field(Name:SWECmd).Value</ApplyTo>
```

```
<ApplyTo>Form.Field(Name:sid).Usage</ApplyTo>
<ApplyTo>Form.Field(Name:sid).Encoding</ApplyTo>
<ApplyTo>Form.Field(Index:0).Name</ApplyTo>
<ApplyTo>Form.Field(Index:2).Value</ApplyTo>
<ApplyTo>Form.Field(Value:Submit with GET).Name</ApplyTo>
```

Offset Length

Value	Description
Type	Signed Numbers
Default value	0
Description	`Offset` and `Length` can be used to target subsets of data referenced with `ApplyTo`.

First, Offset is applied

If Offset >= 0, `Offset` specifies the number of bytes that are to be removed from the beginning of the data.

If Offset < 0, `-Offset` specifies the number of bytes that are to be kept, counting backward from the last byte.

Second, Length is applied

If Length > 0, `Length` specifies the number of bytes that are to be used, counting forward from the first byte.

If Length <= 0, `-Length` specifies the number of bytes that are to be removed, counting backward from the last byte.

`Offset` and `Length` calculations may not be possible, or may be only partially possible.

Example 1

`ApplyTo` returns 30 Bytes of data, `Offset` specifies "40" => Offset calculation is not possible.

In this case the resulting data has the value Null.

Example 2

`ApplyTo` returns 30 Bytes of data, `Length` specifies "35" => Length calculation is possible, however the length must be adjusted to the maximum available value of "30."

In this case the resulting data may have the special value Null or it may equal the maximum data available, depending on the condition attribute `UseDataAvail`.

UseDataAvail

Value	Description
Type	Boolean Values
Default value	true
Description	This attribute specifies what data is to be used when an Offset/Length calculation is only partially possible. If true, the maximum data available is used. If false, the result of the Offset/Length calculation is Null.

IfNull

Value	Description
Type	Extended Boolean Values
Default value	unknown
Description	This attribute specifies the return value of the condition if there is no data to operate on (that is, fi calculations with `ApplyTo`, `Offset`, `Length` and `UseDataAvail` return Null).

Saving Temporary Variables

Conditions which apply to a block of data can specify this data as described in Specifying Data for Conditions.

This block of data can now be saved to a "temporary variable" so that it can be referenced by the `ApplyTo` attribute of a subsequent condition.

The new attributes of such conditions are:

- `SaveAs`
- `SaveIfTrue`
- `SaveIfUnknown`
- `SaveIfFalse`
- `SaveMode`
- `SaveTag`

SaveAs

Value	Description
Type	Strings
Description	This attribute causes the data to be saved and specifies the name of the temporary variable to which the data is saved to.

SaveIfTrue SaveIfUnknown SaveIfFalse

Value	Description
Type	Strings
Default values	• SaveIfTrue: true • SaveIfUnknown: false • SaveIfFalse: false
Description	This attribute specify if the data should be saved in case the result of the condition is true, unknown, or false.

SaveMode

Value	Description
Type	Distinct Values

Value	Description
Default value	IfNew
Description	This attribute specifies how to save the data to the temporary variable.

Allowed values:

- `IfNew`: Saves the data only if there is no value associated with the given variable name yet.
- `Replace`: Always saves the data, replaces an existing value.
- `AppendSeparator`: Appends the data to the existing value if the existing value is not empty, no action otherwise.
- `Append`: Appends the data to the existing value.
- `PrpendSeparator`: Prepends the data to the existing value if the existing value is not empty, no action otherwise.
- `Prepend`: Prepends the data to the existing value.

SaveTag

Value	Description
Type	Numbers
Default value	0
Description	This tag is only applicable to the `RegExpr` condition and allows to specify a tag number referencing a tagged sub-expression of the regular expression is saved

SaveWhat

Value	Description
Type	Distinct Values
Default value	Self
Description	A condition extracts a fragment of the entire block of data which is identified by the `ApplyTo` tag. The `SaveWhat` tag allows to specify if the extracted fragment should be saved, or the fragment left or right of the extracted fragment.

Allowed values:

- `Self`: Saves the fragment identified by the condition.
- `Left`: Saves fragment left of Self.
- `Right`: Saves the fragment right of Self.

Example

```
<Conditions>
  <RegExpr>
    <ApplyTo>Http.Final.Response.Body</ApplyTo>
    <Data>Instructions:\([^&lt;]+\)&lt;</Data>
    <SaveAs>Instructions</SaveAs>
    <SaveTag>1</SaveTag>
  </RegExpr>
  <FindData>
    <ApplyTo>Instructions</ApplyTo>
    <Data>Download</Data>
  </FindData>
</Conditions>
```

Condition Evaluation Environment

The condition attribute `ApplyTo` is resolved within an environment that is configured based on the rule type in which the condition is used.

This chapter explains which attribute `ApplyTo` values are allowed for conditions, depending on where conditions are used.

Access to Special Strings

`HttpParsingRule`, `TcpRuleRecvProto` and `TcpRuleRecvUntil` rule types can reference the strings `All`, `Self`, `Left`, `Right` and `Rest`.

Conditions in the `ScriptGen` sections of HTTP parsing rules have additional options for referencing specific data using the `ApplyTo` property:

- `LinkName`: The current link name (only available when scripting a `WebPageLink` function)
- `FormName`: The current form name (only available when scripting a `WebPageSubmit` function)
- `FormFieldName`: The current form field name (only available when scripting a form field name or form field value)
- `FormFieldValue`: The current form field value (only available when scripting a form field name or form field value)
- `TargetFrame`: The current target frame name (only available when scripting a `WebPageLink` or `WebPageSubmit(Bin)` function)

Examples

```
<ApplyTo>All</ApplyTo>
<ApplyTo>Self</ApplyTo>
<ApplyTo>FormFieldName</ApplyTo>
```

Access to HTTP request/response pairs

Conditions used within `HttpParsingRule` rules can access details of HTTP requests / responses with the `ApplyTo` property.

```
"Http" [ ".Initial" | ".Final" ] ( ".Request" | ".Response" ) [ "."
Component ]
```

The optional component Initial or Final is only significant in cases where HTTP requests are part of a redirection chain. In such cases, Initial returns the first HTTP request in the chain; Final returns the last request in the chain.

Final is the default for conditions in the `HttpParsingRule\Search` section.

Initial is the default for conditions in the `HttpParsingRule\ScriptGen` section.

Valid Component values for HTTP requests, plus return values:

(empty)	Equal to Header
Body	Request body
RequestLine	Request line (i.e., Method + URL + HTTP version)

Method	HTTP method
Version	HTTP version
Header	Complete request header, including request line
Header.*	Use this to reference any HTTP request header
Url	Complete request URL
Url.Complete	Complete request URL
Url.BaseUrl	URL without query string
Url.DirectUrl	Relative request URL (without scheme and host)
Url.BaseDirOnlyUrl	URL without query string and file name
Url.Scheme	URL scheme (HTTP, HTTPS or FTP)
Url.Host	Host name in URL
Url.Port	Port in URL
Url.Path	Path (directory plus file name)
Url.Dir	Directory
Url.File	File name
Url.Ext	File extension
Url.Username	User name
Url.Password	Password
Url.Query	Query string, including "?"
Url.QueryData	Query string, excluding "?"
Url.Coords	Image coordinates

Valid Component values for HTTP responses:

(empty)	Equal to Header
Body	Response body
StatusLine	Response status line (i.e., HTTP version plus status code and status phrase)
Version	HTTP version
StatusCode	HTTP response status code
StatusPhrase	HTTP response status phrase
Header	Complete response header, including status line
Header.*	Use this to reference any HTTP response header

Examples

```
<ApplyTo>Http.Response.Header.Content-Type</ApplyTo>
<ApplyTo>Http.Response.Header</ApplyTo>
<ApplyTo>Http.Request.Url.QueryData</ApplyTo>
<ApplyTo>Http.Request.Body</ApplyTo>
<ApplyTo>Http.Response.StatusCode</ApplyTo>
```

CompareData Condition

The `CompareData` condition checks to see if certain data has a specific value. It uses the attributes `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` to determine what data is subject to the test and what the return value should be if the data isn't available.

Also see section “Specifying Data for Conditions”.

In contrast to other condition types, if the `Length` property is omitted, the number of bytes specified in the `Data` property is used as the `Length` property. By omitting the `Length` property, this allows for a prefix match rather than an exact match.

The Verify condition is an alias for .

Data Attribute

Type	Binary Data
Default	(empty)
Description	The `Data` attribute specifies what data is to be compared to the data referenced by `ApplyTo`, `Offset` and `Length`.

CaseSensitive Attribute

Type	Boolean Values
Default	false
Description	Specifies whether to search for the data in case-sensitive or case-insensitive format.

GenVerify Attribute

This attribute is only applicable if the condition is used within a `TcpRuleRecvProto` rule. See section "GenVerify Attribute Of Conditions" for details.

Example

```
<CompareData>
  <ApplyTo>Http.Response.Header.Content-Type</ApplyTo>
  <Data>text/css</Data>
  <IfNull>true</IfNull>
  <CaseSensitive>false</CaseSensitive>
</CompareData>
```

FindData Condition

The `FindData` condition checks to see if certain data contains a specific value. It uses the attributes `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` to determine what data is to be subject to the test and what the return value should be if the data is not available.

Also see section “Specifying Data for Conditions”.

Data Attribute

Type	Binary Data
Default	(empty)

Description	The `Data` attribute specifies what data is to be found in the data referenced by `ApplyTo`, `Offset` and `Length`.

CaseSensitive Attribute

Type	Boolean Values
Default	false
Description	Specifies whether to search for the data in case-sensitive or case-insensitive format.

IgnoreWhiteSpaces Attribute

Type	Boolean Values
Default	false
Description	Specifies whether or not to ignore white spaces while searching.

GenVerify Attribute

This attribute is only applicable if the condition is used within a `TcpRuleRecvProto` rule. See section "GenVerify Attribute Of Conditions" for details.

Example

```
<FindData>
  <ApplyTo>Http.Request.Url</ApplyTo>
  <Data>/images/</Data>
  <CaseSensitive>true</CaseSensitive>
</FindData>
```

RegExpr Condition

The `RegExpr` condition applies a regular expression to certain data. It uses the properties `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` to determine what data will be subject to the regular expression test and what the return value will be if the data isn't available. Also see section “Specifying Data for Conditions”.

Data Attribute

Type	Binary Data
Default	(empty)
Description	The `Data` attribute specifies the regular expression to be applied to the data referenced by `ApplyTo`, `Offset` and `Length`.

ExpectMatch Attribute

Type	Distinct Values
Default	Any

Description	Specifies the regular expression must match the complete data or if any substring matching the regular expression is sufficient. Allowed values are: • Any: Matching any substring is sufficient. • Complete: Complete data must match the regular expression.

GenVerify Attribute

This attribute is only applicable if the condition is used within a `TcpRuleRecvProto` rule. See section "GenVerify Attribute Of Conditions" for details.

Example

```
<RegExpr>
  <ApplyTo>Http.Response.Body</ApplyTo>
  <Data>&lt;html&gt;.*&lt/html&gt;</Data>
</RegExpr>
```

CheckRange Condition

The `CheckRange` condition checks to see if a numeric value lies within a given range. It uses the properties `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` to determine what data is to be converted to a numeric value and what the return value should be if the data isn't available or the conversion fails. Also see section “Specifying Data for Conditions”.

Range Attribute

Type	Numeric Ranges
Default	-
Description	Specifies the allowed range for the numeric value.

Convert Attribute

Type	Distinct Values
Default	`Length`
Description	Specifies how to convert the data (obtained through the properties `Apply`, `Offset`, `Length`, and `UseDataAvail`) into a number.

Allowed values are:

Length (default)	The data is converted to a number by measuring the length (number of bytes) of the data.
Parse	The data is converted to a number by assuming a textual numeric value and parsing it. If parsing fails, no range check is performed and the attribute IfNull determines the result of the condition.

LittleEndian	The data is treated as a binary representation of a number in little endian format. If the data is empty or is longer than 4 bytes, this fails and the attribute IfNull determines the result of the condition.
BigEndian	The data is treated as a binary representation of a number in big endian format. If the data is empty or is longer than 4 bytes, this leads to a failure and the attribute IfNull determines the result of the condition.

Example

```
<CheckRange>
  <ApplyTo>Http.Response.StatusCode</ApplyTo>
  <Range>200-299</Range>
  <Convert>Parse</Convert>
</CheckRange>
```

ResultLen Condition

The `ResultLen` condition is a special form of the `CheckRange` condition, with two differences:

1. The default value for the `ApplyTo` attribute is not `Self`, but `All`.
2. The accepted range is not specified in the `Range` attribute, but directly within the `ResultLen` - tag.

Example

```
<ResultLen>5-15</ResultLen>
```

NoBlockSplit Condition

The `NoBlockSplit` condition is only applicable within `TcpRuleRecvProto` and `TcpRuleRecvUntil` rules.

It uses the attributes `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` to determine what data is to be subject to the test and what the return value should be if the data isn't available.

Also see section “Specifying Data for Conditions”.

In contrast to other conditions, the default value for the `ApplyTo` attribute is `All`, rather than `Self`.

Semantics

When recording TCP/IP traffic, servers may send responses in multiple TCP/IP packets (blocks). The `NoBlockSplit` condition tests to see if the end of the data obtained by `ApplyTo`, `Offset` and `Length` is identical to the end of one of the packets.

The result is false if the end of the tested data is not on a block boundary; otherwise the result is true.

Example

```
<NoBlockSplit></NoBlockSplit>

<NoBlockSplit>
  <ApplyTo>Left<ApplyTo>
</NoBlockSplit>
```

Scripting Condition

The `Scripting` condition does not operate on specific data, therefore the attributes `ApplyTo`, `Offset`, `Length`, `UseDataAvail` and `IfNull` are not available.

Attribute

Type	Distinct Values Lists
Default	`All`
Description	The attribute value is specified directly within the `Scripting` tag, not within a sub node of the condition.

Allowed values are:

All	An abbreviation for the complete list of other values
Url	Scripting any URL parameter
LinkName	Scripting a link name of `WebPageLink`
FormName	Scripting a form name of `WebPageSubmit`
PostedData	Scripting binary posted data (for example `WebPagePost`)
FormFieldName	Scripting a form field name in the dclform section
FormFieldValue	Scripting a form field value in the dclform section
SetCookie	Scripting the first parameter of the function `WebCookieSet`

`True` is returned if the condition is evaluated while scripting a string parameter in one of the specified script locations, otherwise it's `false`.

Examples

```
<Scripting>Url, FormFieldValue</Scripting>
<Scripting>All</Scripting>
<Scripting>LinkName, FormName, PostedData</Scripting>
```

Exists Condition

The condition type `Exists` checks if the data specified in the `ApplyTo` tag is not null. It is especially useful in conjunction with the `SaveAs` condition tag.

Example

```
<Conditions>
  <Or>
    <Exists>
      <ApplyTo>Form.Submit.Field(Name:SWECmd).Value</ApplyTo>
      <SaveAs>Command</SaveAs>
    </Exists
    <Exists>
      <ApplyTo>Form.Submit.Field(Name:SWEMethod).Value</ApplyTo>
      <SaveAs>Command</SaveAs>
    </Exists>
  </Or>
  <Exists>
```

```
    <ApplyTo>Command</ApplyTo>
    <SaveAs>PageName</SaveAs>
    <SaveMode>Append</SaveMode>
  </Exists>
</Conditions>
```

Loop Condition

The new compound condition `Loop` is similar to the condition `And` in that it evaluates its sub-conditions as long as they evaluate to true. However, unlike the `And` condition, it doesn't stop once all conditions have been evaluated. Instead, the `Loop` condition starts over again and repeatedly evaluates its sub-conditions and stops only when a condition evaluates to false. Of course this only makes sense if at least one sub-condition has side-effects by using the `SaveAs` tag, otherwise it would loop forever once all sub-conditions have returned true on the first pass.

Example

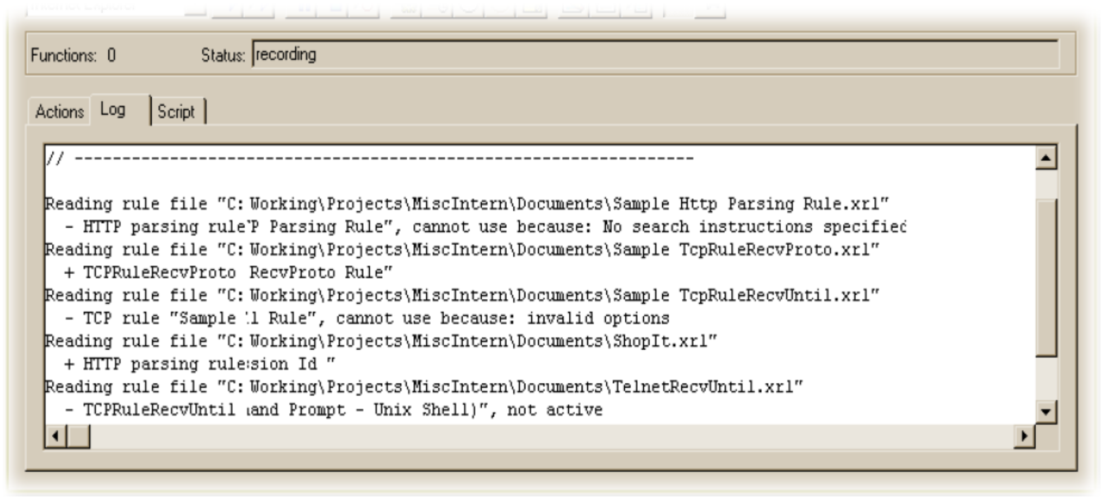
This example is taken from the Flex/AMF3 project type and shows how the page timer name is assembled from the response body.

```
<Conditions>
  <Exists>
    <ApplyTo>Http.Initial.Request.Body</ApplyTo>
    <SaveAs>RestOfBody</SaveAs>
    <SaveMode>Replace</SaveMode>
  </Exists>
  <Loop>
    <RegExpr>
      <ApplyTo>RestOfBody</ApplyTo>
      <Data>operation=&amp;quot;\([^&amp;quot;]*\)</Data>
      <SaveAs>operation</SaveAs>
      <SaveMode>Replace</SaveMode>
      <SaveTag>1</SaveTag>
    </RegExpr>
    <RegExpr>
      <ApplyTo>RestOfBody</ApplyTo>
      <Data>operation=&amp;quot;\([^&amp;quot;]*\)</Data>
      <SaveAs>RestOfBody</SaveAs>
      <SaveMode>Replace</SaveMode>
      <SaveWhat>Right</SaveWhat>
    </RegExpr>
    <Exists>
      <ApplyTo>Literal:, </ApplyTo>
      <SaveAs>OperationList</SaveAs>
      <SaveMode>AppendSeparator</SaveMode>
    </Exists>
    <Loop>
      <FindData>
        <ApplyTo>operation</ApplyTo>
        <Data>.</Data>
        <SaveAs>operation</SaveAs>
        <SaveMode>Replace</SaveMode>
        <SaveWhat>Right</SaveWhat>
      </FindData>
    </Loop>
    <Exists>
      <ApplyTo>operation</ApplyTo>
      <SaveAs>OperationList</SaveAs>
      <SaveMode>Append</SaveMode>
    </Exists>
  </Loop>
  <Exists>
```

```
    <ApplyTo>OperationList</ApplyTo>
    <SaveAs>PageName</SaveAs>
    <SaveMode>Replace</SaveMode>
  </Exists>
</Conditions>
```

Troubleshooting

This section offers some recommendations for troubleshooting recording rules that do not operate as expected. The Recorder outputs diagnostic information to the **Log** tab window, as shown in the screenshot below.



```
// -----------------------------------------------------------------

Reading rule file "C:\Working\Projects\MiscIntern\Documents\Sample Http Parsing Rule.xrl"
  - HTTP parsing rule"P Parsing Rule", cannot use because: No search instructions specified
Reading rule file "C:\Working\Projects\MiscIntern\Documents\Sample TcpRuleRecvProto.xrl"
  + TCPRuleRecvProto  RecvProto Rule"
Reading rule file "C:\Working\Projects\MiscIntern\Documents\Sample TcpRuleRecvUntil.xrl"
  - TCP rule "Sample :l Rule", cannot use because: invalid options
Reading rule file "C:\Working\Projects\MiscIntern\Documents\ShopIt.xrl"
  + HTTP parsing rule:sion Id "
Reading rule file "C:\Working\Projects\MiscIntern\Documents\TelnetRecvUntil.xrl"
  - TCPRuleRecvUntil and Prompt - Unix Shell)", not active
```

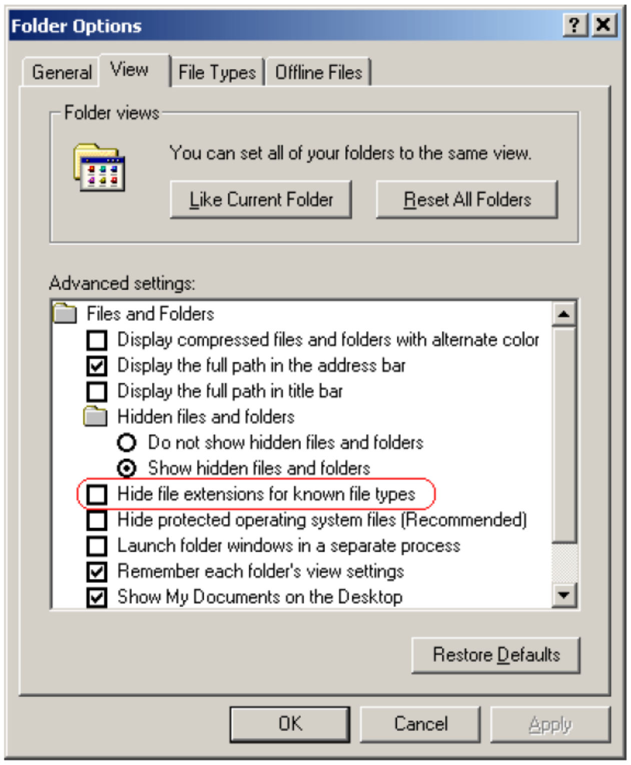
This output contains:

- The recording rule files that have been read by the Recorder.
- The recording rules contained in the recording rule files.
- "+" signs preceding rules that are correct and active.
- "-" signs preceding rules that are incorrect or inactive.
- Reasons why rules aren't used ("not active" or an error description).

If a rule file does not appear in this diagnostic output, check the following:

- Is the rule file in the Documents directory of the current project or the public or user’s RecordingRules directory?
- Does the file have the extension `.xrl`?

Ensure that the setting **Hide file extensions for known file types** is not checked in the Windows Explorer **Folder Options** dialog box, as shown in the screenshot below. Otherwise Windows may show a file name such as `Rule.xrl` when the file name is actually `Rule.xrl.txt`.



You can set all of your folders to the same view.

Advanced settings:

- Files and Folders
  - [ ] Display compressed files and folders with alternate color
  - [✓] Display the full path in the address bar
  - [ ] Display the full path in title bar
  - Hidden files and folders
    - ○ Do not show hidden files and folders
    - ● Show hidden files and folders
  - [ ] Hide file extensions for known file types
  - [ ] Hide protected operating system files (Recommended)
  - [ ] Launch folder windows in a separate process
  - [✓] Remember each folder's view settings
  - [✓] Show My Documents on the Desktop

Manually Written Test Scripts

If you do not want to use the Silk Performer Recorder to generate test scripts, you have the option of creating them manually.

Silk Performer test scripts are written in the program's proprietary scripting language, the Benchmark Description Language (BDL), which is a high-level language that resembles Pascal. To edit scripts manually, you must be familiar with BDL and be able to create prototypes of user requests. You must also be able to define the typical components of a test script, including modules, functions, workload definitions, transactions, and Web forms.

Writing Test Scripts Manually

1. Click **File** > **New** > **Plain Script (.bdf)** . The **Save As** dialog box appears.
2. Give the script a name and click **Save**.

Use Benchmark Description Language (BDL), Silk Performer's proprietary scripting language to write your script.

Editor Keyboard Shortcuts

The script editor provides a number of keyboard shortcuts and keyboard mouse combinations that facilitate the scripting work.

press and hold **Shift+Alt** and drag with mouse	Vertical selection. This lets you edit several rows at once. You can also use the arrow keys instead of the mouse to expand the selection.
press and hold **Alt** and press the arrow up/arrow down key	Move the current line/selected lines up and down.
press and hold **Ctrl** and click an identifier	Takes you to the definition of the identifier (*go to jump*).

Ctrl+Minus	Takes you back to your previous position. Particularly useful after a *go to jump*.
Ctrl+Space	Opens the code completion box.
Ctrl+i	Shows the parameter information box.
double-click the vertical line of a group	Collapses the group.
Ctrl+Shift+L	Deletes the current line.
Ctrl+Enter	Creates a new line above the current line.
Ctrl+U	Converts the selected text to lower case.
Ctrl+Shift+U	Converts the selected text to upper case.
Tab	Increments the indentation.
Shift+Tab	Decrements the indentation.
Ctrl+Q	Comments the selected lines.
Ctrl+W	Uncomments the selected lines.
Ctrl+G	Go to line. Lets you jump to a specific line in the script.

Sample Scripts

Sample script reuse is a timesaving possibility for those who want to generate Silk Performer test scripts manually (and bypass the standard method, which uses the Silk Performer Recorder).

Sample project files, including scripts, include files, data files, profile files, workload definitions, and more, are available from the **Type** list on the **Outline Project** dialog when you are initially prompted to outline a new project. These files can be used as templates that you can edit manually to meet your needs.

Silk Performer test scripts are written in the program's proprietary scripting language, the Benchmark Description Language (BDL), which is a high-level language that resembles Pascal. To edit scripts manually, you must be familiar with BDL and be able to create prototypes of user requests. You must also be able to define the typical components of a test script, including modules, functions, workload definitions, transactions, and Web forms.

Working From Sample Scripts

1. Open the project in which you want to work, or start a new project.
2. Click **File** > **Open** > **Script File (.bdf)**. The **Open** dialog box appears.
3. Browse to the **Samples** folder (`<public user documents>\Silk Performer 20.0\Samples\<...>`). Select a sample script and click **Open**.
4. Confirm that you want to add the script to your project.

Trying Out Scripts

Silk Performer offers several means of evaluating test scripts, starting with the execution of a Try Script run.

Try Script Overview

Part of the process of conducting a Silk Performer load test is to perform a trial run of the test script that was created during script modeling.

Normally, this is traffic recorded by the Silk Performer Recorder during script modeling. For a trial run, or Try Script, of a test script, options are automatically selected so that you can see a live display of the actual data downloaded. Log files, TrueLog files, report files, output files, and error files are created so that you can later confirm that the script works properly. Only one user is run, and the stress test option is enabled so that there is no think time and no delay between transactions. However, at this stage, the measurements typical of a real load test are not performed.

Try Script Settings

For Try Script runs, the following options are automatically set to these specified values (see also "Replay Options"):

- The **Stress test** option is on, when think times are disabled.
- The **Stop virtual users after simulation time (Queuing Workload)** option is off.
- The **Virtual user log files (.log)** option is on.
- The **Virtual user output files (.wrt)** option is on.
- The **Virtual user report files (.rpt)** option is on.
- The **Virtual user report on error files (.rpt)** option is on.
- The **TrueLog files (.xlg)** option is on.
- The **TrueLog On Error files (.xlg)** option is off.
- The **Compute time series data (.tsd)** option is off.
- All logging detail options (**Results** > **Logging** and **Results** > **Internet Logging** page) are on.
- The **Enable all measure groups (TSD measure groups)** option is off.
- The **Bandwidth** option is set to `High Speed (unlimited)`.
- The **Downstream** option is set to `unlimited`.
- The **Upstream** option is set to `unlimited`
- The **Duplex** option is off.

Trying Out a Test Script

You must record or manually create a test script before you can run a Try Script.

1. Click the **Try Script** button on the Silk Performer Workflow bar. The **Workflow – Try Script** dialog appears.
2. Choose a script from the **Script** list box.
3. In the **Profile** list box, the currently active profile is selected (this is the default profile if you have not configured an alternate profile).
   a) To configure simulation settings for the selected profile, click **Settings** to the right of the list box.
   b) To configure project attributes, select the **Project Attributes** link.
4. In the **Usergroup** list of user groups and virtual users, select the user group from which you want to run a virtual user.

   Since this is a Try Script run, only one virtual user will be run.
5. To view the actual data that is downloaded from the Web server during the Try Script in real-time, select the **Animated Run with TrueLog Explorer** check box.

   If you are testing anything other than a Web application, you should disable this option.
6. Click **Run**. The Try Script begins.

All recorded think times are ignored during Try Script runs. The **Monitor** window opens, giving you detailed information about the progress of the Try Script run. If you have selected the **Animated** option, TrueLog Explorer opens. Here you can view the actual data that is downloaded during the Try Script run. If any errors occur during the Try Script run, TrueLog Explorer can help you to find the errors quickly and to customize session information. Once you have finished examining and customizing your script with TrueLog Explorer, your script should run without error.

Try Script Summary

When a Try Script run is complete, the **Try Script Summary** page appears. You can also open this page from the **Results** tree. You can perform the following actions in the **Next Steps** area on the right side:

- Click **Customize user data** to open the TrueLog Explorer and customize your script.
- Click **Define User Types** to continue with the next step in the workflow bar. This button is only visible if you use the **Simple Workflow Bar**.

- Click **Find Baseline** to continue with the next step in the workflow bar. This button is only visible if you use the **Full Workflow Bar**.

In the **Analyze Result Files** area, you can open various reports and log files.

Note: If you want to prevent the summary page to appear each time a test is complete, disable the **Show Summary Page** button in the toolbar of the **Monitor** page.

Visual Analysis with TrueLog Explorer

TrueLog Explorer guides you through the process of visually analyzing Try Script runs with its Workflow bar.

TrueLog Explorer provides an accurate view of the actual data that is downloaded during tests. This enables you to verify that the data specified to be received from the server under test was, in fact, received. In the case of Web application testing for example, TrueLog Explorer shows the actual Web pages that are received during tests. Through the use of TrueLog Explorer's animated mode, live monitoring of downloaded data is available—data is displayed as it is received during tests.

TrueLog Explorer provides different views of received data, enabling you to:

- Inspect the rendered HTML code, as a real user would see it.
- Inspect the native HTML code.
- Inspect a difference table that shows the differences between replay and recorded TrueLogs.

Validating Scripts with TrueLog Explorer

Run a script before performing this task.

1. From the Silk Performer menu bar, select **Results** > **Explore TrueLog** .

   Note: Selecting the **Animated Run with TrueLog Explorer** check box before running a Try Script operation opens TrueLog Explorer automatically.
2. Examine the Virtual User Summary report.

   A Virtual User Report is shown automatically when the first node of the menu tree at the left side of the window is checked.

   The report shows detailed information about the Try Script run in tabular form.
3. Use the menu tree on the left of the window to expand and collapse folders as necessary to locate and view data downloaded during the test.
4. To view a downloaded Web page, select its URL or link description in the menu tree.
5. To locate replay errors, click the **Analyze Test** button on the workflow bar and select **Find Errors**. A new window appears that helps you navigate from one error to the next. HTML errors often occur one or two steps before they are apparent to end users. With TrueLog Explorer you can easily examine the Web pages that were displayed preceding errors.
6. Compare a replay TrueLog with the TrueLog that was produced during recording of the application.

Log Files

If log-file generation has been enabled, then any errors that are encountered in load tests are reported in log files. Log files contain a record of all the function calls that are invoked by the transactions of each virtual user in load tests. These files contain session and error information, including details for each type of event executed during a simulation, the API calls in a transaction, and the data exchanged with the server.

The process of generating log files alters the time measurements of tests. Therefore, use log files for debugging purposes only. Do not generate them for full load tests.

Viewing Log Files

Choose one of the following:

- From the Silk Performer menu bar, select **File** > **Open** > **Virtual User Log File** and then locate the appropriate file (`.log`).
- Right-click the line for a virtual user in the Silk Performer Monitor window and select **Show Virtual User Log File** from the context menu.

The log file opens.

Viewing Report Files

Report files contain the same information as baseline reports for one specific virtual user. This report is based on XML/XSL and includes the most important test results in tabular format.

Note: When TrueLog Explorer opens, the Virtual User Report file is displayed automatically.

Choose one of the following:

- From the Silk Performer menu bar, select **File** > **Open** > **Virtual User Report File** and then locate the appropriate file (`.rpt`).
- Right-click the line for a virtual user in the Silk Performer Monitor window and select **Show Virtual User Report File** from the context menu.

The report file opens.

Report Contents

A report file contains the following sections:

- General Information
- Virtual User
- Summary Tables
- Transaction Response Times
- HTML Page and Action Timers
- Web Form Measurements

General information

The general information section includes administrative information in a tabular form.

Administrative information includes the Silk Performer version info, the project name, a description of the project, the date and time of the test run, the workload definition, the workload model and the number of errors that occurred.

User types

For the virtual user, a section is available with details of the measured response times. The summary line shows the duration of the test, the session time, the session busy time, the average page time, the average action time, the number of transactions executed successfully, canceled, failed and the number of errors, if any, that occurred.

Summary tables

This section contains summary measurements in a tabular form, that is, aggregate measurements. The first table provides general information, such as the number of transactions that were executed and the number of errors, if any, that occurred. All the following tables provide summary information relevant to the type of application that was tested.

Transactions

This section contains summary measurements in a tabular form, that is, aggregate measurements for all transactions of the specific user. For every transaction, the transaction response time and the transaction busy time are displayed.

The transaction response time is measured from the beginning to the end of the transaction, including all think times.

The transaction busy time is the transaction response time without any think time included.

Page and action timers

This section contains summary measurements in a tabular form for every accessed Web page. For every page, the download time of the whole page, the HTML-document download time, the server busy time, the amount of data downloaded with this page, and the amount of data downloaded for embedded objects are displayed.

Web forms

This section contains summary measurements in tabular form for every used Web form. For every form, the round trip time, the server busy time, the number of HTTP hits, the amount of request data sent and the response data received are displayed.

Output Files

Output files contain the output of write statements that are used in test scripts. An output file is generated for a particular user only if write statements are executed by that user.

Note: Generating output files alters the time measurements of load tests. Therefore, these files should be used for debugging purposes only, and should not be generated for full load tests.

Viewing Output Files

Note: Generating output files alters the time measurements of load tests. Therefore, these files should be used for debugging purposes only, and should not be generated for full load tests.

Choose one of the following:

- From the Silk Performer menu bar, select **File** > **Open** > **Virtual User Output File** and then locate the appropriate file (`.wrt`).
- Right-click the line for a virtual user in the Silk Performer Monitor window and select **Show Virtual User Output File**.

The output file opens.

Customizing Scripts

After you have recorded and tried out your script for the first time, you may need to customize it for two reasons:

- The script does not replay without errors.

  Session-specific data that was captured during recording is usually not valid during replay. This is because data like session IDs change every time a new communication session is started with the

application under test. Session-specific data can be represented by variables and is, consequently, dynamically adjusted during each run. This leads to clean scripts that work flawlessly every time they are executed.
- You want to introduce more variety in virtual user behavior.

  The virtual users will behave identically during your load test, unless you introduce data or environmental variation, such as simulating different browsers, bandwidths, or paths through the application.

Therefore, it is recommended to customize your scripts before you start a load test. Customizing scripts offers the following advantages:

- Your virtual users will be diverse.

  For example, you can set them up to use different browsers and bandwidths from different locations to access the application under test.
- Your virtual users will behave more realistically and more natural.

  This can be achieved with randomized data. The virtual users will then, for example, input varying data in forms, like different names, addresses, and credit card numbers. They can vary the product names they search for, the products they order, and so on. This leads to accurate and meaningful result data.

Customizing scripts is typically done before you move on with the next step in the workflow bar.

Function and Transaction Declarations

Silk Performer offers GUI tools that assist you in inserting function and transaction declarations into test scripts.

A function is used to encapsulate some tasks that a script requires. Functions can access and change data declared outside themselves, either through variables or parameters. Optionally, a function can return a value of a specific type.

A transaction is a discrete piece of work that is assigned to a virtual user in a test and for which separate time measurements are made.

Inserting Function Declarations

1. With the test script into which you want to insert a new function open, select **Script** > **New Function** from the Silk Performer menu bar. You can also access this dialog by right-clicking within the body of the script and selecting **New Function** from the context menu. The **New Function** dialog appears.
2. In the **Name** field, enter a name for the function you want to insert into your test script. The name must begin with a character and may consist of any number of letters of the alphabet, digits, and underscores.
3. From the **Return type** list box, select the data type of the function's return value. Choose one of the following:
   - `none` - for the function to return nothing
   - `boolean` - for the function to return either true or false
   - `float` - for the function to return any positive or negative floating-point value or zero
   - `number` - for the function to return any positive or negative whole number or zero
   - `string` - for the function to return a sequence of characters
4. In the **Parameters** list, insert the parameters that you want to pass to the function. For each parameter, enter a name in the **Name** column and select the data type from the **Type** list box:
   - `boolean` - for the parameter to contain either true or false
   - `float` - for the parameter to contain any positive or negative floating-point value or zero
   - `number` - for the parameter to contain any positive or negative whole number or zero

- `string` - for the parameter to contain a sequence of characters

5. Use the **Preview** area to see how the function declaration looks when it is inserted into your script. The declaration contains the parameters you insert into the list as well as the return type. Keep in mind that it is up to you to add the function code after the function declaration has been inserted into the script.
6. Click **OK**.

The new function is placed in your script, and appears in the script editor window.

Inserting Transaction Declarations

1. With the test script into which you want to insert a new transaction open, select **Script** > **New Transaction** from the Silk Performer menu bar. You can also access this dialog by right-clicking within the body of the script and selecting **New Transaction** from the context menu. The **New Transaction** dialog appears.
2. In the **Name** field, enter a name for the transaction you want to insert into your test script. The name must begin with a character and may consist of any number of letters of the alphabet, digits, and underscores.
3. Click **OK**.

The new transaction is placed in your script, and appears in the script editor window.

User Groups

User groups are groups of virtual users who share similar, though not identical, behavior. The members of user groups perform slightly different transactions with the goal of simulating real-world client behavior.

User groups are used in conjunction with parameterized user input data and profiles, for example browser types and connection speeds, to simulate typical end-user activity.

User groups are defined by the *transactions* that they call and the frequency at which they call those transactions. For example, you might have a user group called `Searcher` that performs a `TLogon` transaction once, a `TSearch` transaction five times, and a `TLogoff` transaction once. You might also have a user group called `Updater` that performs the `TLogon` transaction once, the `TSearch` transaction three times, the `TUpdate` transaction twice, and the `TLogoff` transaction once. A third user group called `Inserter` might perform the `TLogon` transaction once, the `TSearch` transaction once, the `TInsert` transaction three times, and the `TLogoff` transaction once. Collectively, these user groups can be used to simulate real-world client activity while maintaining load balance, meaning that transactions are distributed in such a way that all users don't perform the same transaction at the same moment.

When multiple users have the same profile, they can be combined into a user group. The user group list contains the name of the user group, the transactions that are to be invoked by the user group, and the frequency at which the transactions are to be invoked (`TransCount`). User group descriptions are included in the `dcluser` sections of Silk Performer test scripts.

Customizing User Groups

1. Open and activate the BDF script you want to customize
2. Select **Script** > **Customize User Groups** from the Silk Performer menu bar. The **Customize User Groups** dialog appears.
3. Right-click a user group or transaction node to open a context menu that lists the available operations:
   - Add a new user group
   - Copy a user group
   - Rename a user group
   - Delete a user group
   - Insert a transaction

- Copy a transaction
- Rename a transaction
- Remove a transaction

   Transactions can be added and removed for each user group. The transaction renaming process renames transactions in all user groups to which they have been assigned.
4. Select the desired action from the context menu and follow the prompts
5. `Begin` and `End` transactions can be defined for each user group by selecting/deselecting the check boxes in the **Begin/End** column.
6. Entries in the **Count** column, which represent the number of times that transactions are to be called, can be edited directly by clicking them.
7. When the **Choose transactions randomly** check box is selected and a user group contains at least two transactions that are neither "begin" transactions nor "end" transactions, the **Random Order** column shows how the transaction distribution will be weighted, by percentage. For example, if two non-begin and non-end transactions are present, each transaction will have a random order distribution of 50%. If three non-begin and non-end transactions are present, the transactions will have random order distributions of 33%, 33%, and 34%.
8. Click **OK** when you've completed user group and transaction customization.

The modified data is written to the `dcluser` section (and `dcltrans` section if necessary) of your test script.

Creating Parameters with the Parameter Wizard

The Parameter Wizard enables realistic functionality to be created for virtual users during tests by replacing recorded or scripted values with constant values, random values, or values pulled from multi-column data files. Each of these value types can be generated using the wizard. This means that you can furnish virtual users with varied personal data for all the information commonly required by Web forms (for example, names, addresses, zip codes, or credit card numbers).

The Parameter Wizard guides you through the process of creating variables and then places the variables in test scripts. A wide range of variable types are provided. Each variable type can generate different value types (strings containing text and/or numbers, serial numbers, whole numbers, and floating-point numbers).

Values can be randomized using different distribution patterns. Comprehensive sets of pre-defined data (for example, names, addresses, or passwords) are also provided.

Inserting a Constant Value Into a Script

1. In the menu bar, click **Script** > **Create New Parameter** . The **Parameter Wizard - Create New Parameter** dialog box appears.
2. Click the **Create New Parameter** icon to open the Parameter Wizard.
3. Select **Constant value**.
4. Click **Next**.
5. Select the data type of the variable that you want to insert into your test script from the **Data type** list box.
6. In the **Name** field, enter a name for the new constant value or accept the default name.
7. In the **Value** field, enter the value of the new constant value.
8. Click **Finish**. The new constant is added to the parameter section of your script. If you opened the wizard via the context menu in your script, the value you selected will be automatically replaced with the newly created variable.

Launch the wizard as often as is required to create more constant values. If you launched the wizard via another method, replace the constant values in your script with the BDL variables you created.

Inserting Random Values Into a Script

1. In the menu bar, click **Script** > **Create New Parameter** . The **Parameter Wizard - Create New Parameter** dialog box appears.
2. Select **Parameter from Random Value**.
3. Click **Next**.
4. In the wizard's **Choose the type of random variable** window, choose a random variable type from the **Random type** list box.

   Use the type description and variable declaration preview provided for each random variable type to choose the required variable and specify its attributes.
5. Click **Next**. The **Name the variable and specify its attributes** window appears.
6. In the **Name** field, enter a name for the new constant value or accept the default name.
7. Specify the attributes of the variable. These vary depending on the type of variable selected. Use the **Back** button to review the type description of the variable if necessary.
8. Click **Next**. The **Choose the kind of usage** window appears.
9. Specify how the variable is to be utilized.

   - `Per usage:` Each use of the random variable generates a new random value.
   - `Per transaction:` A new value is generated for each transaction call.
   - `Per test:` The value is the same for the entire test.
10. Click **Finish**. The new random variable is then added to the random variables section of your script.

    If you opened the wizard via the context menu in your script, the value you selected will be automatically replaced with the newly created variable.

Launch the wizard as often as is required to create more randomized variables. If you launched the wizard via another method, replace the constant values in your script with the BDL variables you created.

Inserting Parameters with Values from Multi-Column Data Files

1. In the menu bar, click **Script** > **Create New Parameter** . The **Parameter Wizard - Create New Parameter** dialog box appears.
2. Select **Parameter from multi-column data file**.
3. Click **Next**.
4. In the wizard's **Parameter from multi-column data file** window, choose a multi-column data file from the **File Name** list box.

   a) Select an appropriate separator from the **Separator** list box, or type a custom separator.
5. In the **Handle name** field, type a name for the file handle of the multi-column data file variable or accept the default name.
6. In the **Parameter name** field, type a name for the new parameter or accept the default name.
7. Click **Next**. The **Parameter Wizard - Parameter from Multi-column Data File** window appears.
8. Select a **Row selection order** and specify the **Attribute** options.
9. Specify the attributes of the variable. These vary depending on the type of variable selected. If necessary, use the **Back** button to review the type description of the variable.
10. Click **Finish**.

    The new parameter from a multi-column data file is then added to the parameter section of your script.

    The multi-column data file you selected is also added to your project's data files.

    If you opened the wizard via the context menu in your script, the value you selected is automatically replaced with the newly created variable.

Launch the wizard as often as is required to create more randomized variables. If you launched the wizard via another method, replace the constant values in your script with the BDL variables you created.

Editing Multi-Column Data Files

1. Open the **Data Files** sub-tree of the **Project** menu tree.
2. Double-click the multi-column data file you want to edit. The **Edit CSV File** window appears.
3. Choose an appropriate separator for the multi-column data file from the **Separator** list box or enter a custom separator.
4. Modify, add, or delete values from the multi-column data files as required.
5. Click **OK** to save the changes of the multi-column data file and close the window, or click **Save As** to save the multi-column data file with a different name.

Inserting a Parameter into Your Test Script

1. Right-click a customizable value (for example, a string) in your BDL script.
2. Select **Customize Value** from the context menu.

   The Parameter Wizard appears.

   The **Customize Value** menu item is disabled if the value in your script is not customizable.
3. In the wizard's **Customize Value** window, select **Use existing parameter**.
4. Click **Next**. The wizard's **Use Existing Parameter** window appears.
5. Select a variable from the list and click **Finish**.

   You can choose the data type of the shown parameters from the **Show all Parameters of type** list box.

   Silk Performer automatically shows the window with a selected data type that is compatible with the data type of the customized value.

The value you selected is automatically replaced with the existing parameter.

Serialization and Synchronization

There are several Silk Performer functions that allow you to either serialize or synchronize transactions in your test scripts to better control the timing of loads in your simulation. Both serialization and synchronization are implemented using high-level functions that call pre-existing Windows functions.

Serialization

Serialization enables you to test the effects of transactions executed one at a time. This is useful, for instance in connecting to a server that is not able to handle a large number of concurrent logins.

Serializing users and transactions requires the use of a token or mutex object. Named mutex objects are created by calling the CreateMutex function. The users to be synchronized have to wait until they receive the mutex object. This guarantees that only one user executes a transaction at a time.

After a user has completed the transaction, the user must release the token by calling the ReleaseMutex function. This makes the mutex object available for other users.

Serialization Functions

The following functions are used in serializing transactions and users:

- `CreateMutex`
- `CreateMutexEx`
- `ReleaseMutex`

- `WaitForSingleObject`
- `CloseHandle`

Serialization Sample Script

The following example is excerpted from a sample script called `Mutexlogin.bdf`, located in the `Samples\Database` folder in the Silk Performer installation directory. The script consists of three transactions. `TMain` waits until the user possesses the token, and then connects to the DBMS; `Selling` executes the database transaction; and `CleanUp` closes the connection to the database and releases all the resources.

Initially, a mutex object called `MyConnectMutex` is created to serialize access to critical sections inside transactions for multiple concurrent users. Calling the `WaitForSingleObject` function effects a delay until the mutex object is in the signaled state. This means that from that point forward, only the current transaction is executed, and this condition remains until the token is released. During execution time a connection to a database system is established. After successfully connecting, the ownership of the mutex object is released. After these steps have been completed successfully, the transaction is committed. In this example, only connecting to the database system is serialized.

```
benchmark MutexLogin
use "dbapi.bdh"
var
  gHdbc1 : number; // handle for database connection
  gHMutex : number; // mutex handle
dcluser
  user
    Seller
  transactions
    TMain : begin;
    Selling : 50;
    CleanUp : end;

dcltrans
  transaction TMain
  var
    str: string;
  begin
    // SYNC: serialize connect transaction with mutex object
    // (enter critical section)
    gHMutex := CreateMutex("MyConnectMutex");
    WaitForSingleObject(gHMutex, INFINITE);
    gHdbc1 := DB_Connect("dsn=sqs_purple;uid=u1;pwd=u1");
    Print("Connected!", 1, 1);
    // SYNC: leave critical section and pass control to other
user
    ReleaseMutex(gHMutex);
  end TMain;

  // process database transaction
  transaction Selling
  begin
  ...
  end Selling;

  transaction CleanUp
  begin
    DB_Disconnect(gHdbc1);
  end CleanUp;
```

Synchronization

Synchronization is useful in testing the effects of a large number of simultaneously called transactions. It allows you to create stress on your system to test specific concurrency issues. Synchronization works by employing an additional user called an *Event Starter*, which functions like the starting gun at the beginning of a race. Using the `CreateEvent` function, you can use the Event Starter user to send an Event object after a specified period of time. You can set up the users in the simulation to wait for this event and thus perform a specified transaction simultaneously when the event is released. Silk Performer provides the `PulseEvent` function for this purpose.

The other users call the `WaitForSingleObject` function to wait until the specified event is in the signaled state. Using the `WaitFor` function, users are able to synchronize themselves without the use of event objects.

Synchronization Functions

The following functions are used in serializing synchronizing and users:

- `CreateEvent`
- `PulseEvent`
- `WaitFor`
- `WaitForSingleObject`
- `CloseHandle`

Synchronization Sample Script

In the following example, the users create an event object called `StarterEvent` and establish a connection to an FTP server. The Starter user sets the state of the event object `StarterEvent` to signaled every ten seconds and resets it after the appropriate number of waiting users process the transaction. In this way, all the Uploader users are forced to execute the `TUpload` transaction at the same time. The Uploader's wait until the event object is in the signaled state, and then execute their Upload transaction.

```
dcluser
  user
    Starter
  transactions
    TInit : begin;
    TStartEvent : 20;
  user
    Uploader
  transactions
    TInit : begin;
    TUpload : 20;
var
  hEventGo : number; // handle for starting event
  hFTP : number;

dcltrans
  transaction TInit
  begin
    hEventGo := CreateEvent("StarterEvent");
    WebFtpConnect(hFTP, "standardhost", WEB_PORT_FTP,
"username", "password");
  end TInit;

  transaction TStartEvent
  begin
    wait 10.0;
    // every 10 seconds the starter event forces all
    // seller users to execute the selling transaction
    // at the same time
    PulseEvent(hEventGo);
  end TStartEvent;

  transaction TUpload
  begin
    WaitForSingleObject(hEventGo, INFINITE);
    WebFtpPut(hFTP, 4096, "test.dat", true);
  end TUpload;
```

Rendezvous Functionality

This function is used to create a delay until the specified virtual users reach a certain state, and then to proceed with all the users simultaneously. This tests the ability of the system to handle a number of concurrent transactions by making it possible to generate very specific stress on the system. For example, in a transaction that stores a number of files on an FTP server, you may want your script to wait until all the users are logged on, and then to store all the files at the same time.

Rendezvous Functions

Silk Performer includes the following rendezvous function:

- `WaitFor`

Rendezvous Sample Script

In this example, the script waits until five virtual users (Buyer) are at the line in the script containing the `WaitFor` function, and it then proceeds with the transaction.

```
const
  USERS := 5;
dcluser
  user
    Buyer
  transactions
    TBuy : 10;

dcltrans
  transaction TBuy
  const
    TIMEOUT := 60;
  begin
    // wait for users rendezvousing at this point
    if not WaitFor("Rendezvous", USERS, TIMEOUT) then
      write("timeout"); halt;
    end;
    // proceed with transaction
    print("rendezvous", 1, TEXT_RED);
  end TBuy;
```

Multiple Agents

The serialization and synchronization functionality is available only on a single agent computer. Silk Performer provides an additional number of functions that extend the use of serialization and synchronization across multiple agents.

Serialization across multiple agents is made possible by the `GlobalResourceEnter` and `GlobalResourceRelease` functions. `GlobalResourceEnter` defines a resource that can be occupied by only a limited number of users. The caller has to wait until the number of users who occupy the resource is less than the maximum number of users who are allowed to occupy the resource. The `GlobalResourceRelease` function releases a resource that was occupied with the `GlobalResourceEnter` function. If there are users still waiting for the resource, the next user in the queue resumes the simulation.

Synchronization across multiple agents is made possible by the `GlobalWaitFor` function. This function defines a checkpoint (rendezvous point) and blocks the calling user. The caller waits until a specified number of users (including the caller) have reached or passed the checkpoint, or until a specified timeout occurs. The checkpoint is identified by name and is visible to all the users, including those running on various remote agents. A waiting user continues when the specified timeout is reached or the given number of users have called the function for a checkpoint. A user who has timed out is also regarded as having passed the checkpoint.

The functions `GlobalVarGet`, `GlobalVarInc`, and `GlobalVarSet` are used, respectively, to get, increment, and set the values of global integer variables that are accessible for all the users on all the agents.

For detailed information on these functions, refer to the BDL Function Reference.

Check list for building robust scripts

A script that was recorded with the Silk Performer Recorder provides a good starting point for a load test. However, such a raw script usually has two main downsides: It will not produce realistic results and it will not be particularly robust. It is likely to cause errors during replay.

To make your scripts as robust as possible, it is recommended to customize them before you start a load test. When customizing your script, consider the following:

- **Appropriate test data**: Most applications require appropriate test data. Make sure to customize your script in a way that makes virtual users employ appropriate user names with correct access rights. If your virtual users execute transactions concurrently, you will probably need to define different user IDs.
- **Dynamic content**: Often, the content of an application is dynamic, like link names that change, items being no longer available, products in a web shop being out of stock, and so on. Also, real users usually pursue different paths through an application. A recorded script, however, only contains one specific path. The path may also vary based on time, user, or other conditions.
- **Not available applications**: When you start your load test, the application under test might not be available, not yet ready, or a database connection is not yet established. Make sure that your script handles these circumstances appropriately.
- **Timing issues**: Some BDL functions use default timeout settings that might not be appropriate for your testing scenario. Adjust these timeout settings to fit your specific needs and to avoid errors.
- **Concurrency issues**: Controlling concurrence issues also helps to avoid errors. Your database might be able to respond to only a few virtual users concurrently. Controlling concurrency also helps to make behavior of virtual users more realistic: Real users usually perform one specific task not all at the same time but in a staggered manner. The serialization, synchronization, and rendezvous functionalities of Silk Performer allow you to precisely control the desired behavior of your virtual users.

To make your script even more robust, also consider the following error scenarios. Reflect about how your virtual users are supposed to act when an error occurs. The virtual users can …

- **skip the remaining part of a transaction**: This action might be appropriate when a virtual user attempts to buy an item in a web shop that is not in stock at that time. The virtual user could skip that particular transaction and continue with executing the next transaction. Such a behavior can be achieved with a return statement.
- **stop the load test**: This action might be appropriate when a database connection cannot be established and the application under test cannot be used without this connection. In such a case, continuing the load test would not be possible. Such a behavior can be achieved with a halt statement.
- **handle an error and continue executing the transaction**: This might be useful when a virtual user enters a wrong password on a login page. If the script could handle such a login error, the virtual user

could attempt to login with a different password and might be able to continue executing the rest of the transaction. Such a behavior can be achieved with error-handling functions.

Custom DLLs

In Silk Performer scripts, it's possible to call functions that are provided in Dynamic Link Libraries (DLLs). This feature allows you to implement functions in any programming language that can generate DLLs and then utilize such external functions in Silk Performer test scripts.

External Function Declaration

Two files are typically required to create a DLL:

- A source file that contains the implementation of one or more functions
- A definition file that specifies which functions are to be exported

When developing a DLL using a common C/C++ compiler, the source file (`.c`, `.cpp`) and the definition file (`.def`) must be included in the Visual Studio project.

In BDL, all external functions called from a script must be declared before they can be used. This must be done in the external functions section of Silk Performer scripts.

Note: If a DLL file is not located in the Silk Performer installation directory or a directory included in the path specification, you need to add the DLL file to the project's data files.

Note: The `vb.net` dll file is not supported.

The following files contain sample scripts. The files can be found in the following folder and its subfolders: `C:\Users\Public\Public Documents\Silk Performer <version>\SampleApps\CustomAPI`.

- `DemoFunctions.bdf`
- `DemoFunctions.dsp`
- `demofunctions.cpp`
- `DemoFunctions.dll`

Script Example

In the following example, the external functions `DemoSum` and `DemoDiff` are declared and then called in the `TMain` transaction.

```
dll "DemoFunctions.dll"

  // function without parameters and return value
  "DemoOutput"
    function Output;

  // function declared using C data types
  "DemoSum"
    function Sum(in long, in long): long;

  // function declared using BDL data types
  "DemoDiff"
    function Diff(in number, in number): number;

// DLL is not located in the Silk Performer directory
dll "C:\\Debug\\Functions.dll"

dcltrans

  transaction TMain
  var
    x, y, nSum, nDiff: number;
  begin
    x := 4; y := 5;
    nSum := Sum(x, y); // external function calls
    nDiff := Diff(x, y);
    Output();
  end TransMain;
```

Function Prototypes

In the external functions section of a test script, after the name of a DLL has been declared, any number of prototypes of DLL functions can be defined. Each function that is to be used in a Silk Performer script must be declared; these declarations are case-sensitive. Additionally, identifiers that specify the names of all functions that are to be used in BDL scripts must be defined.

When parameters are passed to external functions, it is necessary to indicate whether they are passed into functions by reference or by value (default). Parameters can be declared with BDL data types or basic C data types (long, short, char, float, double, or pointer).

The following table shows the data-type mapping between formal parameters of native functions exported by DLLs, formal parameters of external functions declared in the DLL sections of BDH files, and actual parameters (BDL data types).

Table 1: Data-type mapping

C Data Type	DLL Section (BDH)	BDL Data Type
char	char	Number
unsigned char	unsigned char	Number
short	short	Number
unsigned short	unsigned short	Number
long	long, number, boolean	Number, Boolean
unsigned long	long, number	Number
char*	string	String
*(Pointer)	long, number	Number
float	float	Float
double	double	Float
Ptr (C data type representing a pointer to any data type)	string	String

Parameters in External Functions

Any type of parameter can be passed to external functions. While using simple data types is easy, handling arrays and structured data types requires knowledge about the alignment and size of the specific data types in use.

Simple data types allowed as input and output parameters are:

Table 2: Simple data types

C Data Type	BDL Data Type
Char, unsigned char	Number
Short, unsigned short	Number

C Data Type	BDL Data Type
Long	Number
Double	Float
Float	Float
Char*	String

Note: The unsigned long data type is converted to long when used in Silk Performer scripts.

When a parameter is passed to a function by value, it is declared simply by specifying its type. If a parameter is passed by reference (for example, if it is used as an output parameter), a pointer to the data type must be defined.

The example below illustrates how function parameters are declared in C and how corresponding external functions must be declared in BDL. In addition, an example of an external function call is provided.

C Source Code

In this example, the external function `T_PARAM` receives the following input parameters: `inChar`, `inShort`, `inLong`, `inDouble`, `inFloat`, `inUShort`, and `inUChar`. A corresponding output parameter belongs to each: `outChar`, `outShort`, `outLong`, `outDouble`, `outFloat`, `outUShort`, and `outUChar`. These parameters must be declared as pointers because during function calls references to the variables are passed to the functions. Additionally, a string called `inoutString` is declared as an input and output parameter.

```
char* T_PARAM( char            inChar,      // in
               short           inShort,
               long            inLong,
               double          inDouble,
               float           inFloat,
               unsigned short  inUShort,
               unsigned char   inUChar,
               char*           inoutString, // in/out
               char*           outChar,     // out
               short*          outShort,
               long*           outLong,
               double*         outDouble,
               float*          outFloat,
               unsigned short* outUShort,
               unsigned char*  outUChar )
{
  char sBuffer[1000];
  // copy the values of the input parameters to the
  // output parameters and the return value
  return sBuffer;
}
```

BDL External Function Declarations

To use the above-defined `T_PARAM` function in BDL, the function must be declared in the external functions section of a Silk Performer script. This is illustrated in the example below. In this case, the parameters are defined as formal data types. The comments attached to each parameter indicate the corresponding BDL data type.

```
"T_PARAM" function t_param(
  in char,                         // in number
  in short,                        // in number
  in long,                        // in number
  in double,                      // in float
  in float,                       // in float
  in unsigned short,              // in number
  in unsigned char,               // in number
  inout string,                   // inout string
  out char,                       // out number
  out short,                      // out number
  out long,                       // out number
  out double,                     // out float
  out float,                      // out float
  out unsigned short,             // out number
  out unsigned char): string(100); // out number
```

BDL External Function Calls

The following example illustrates how to call the `T_PARAM` function. In this example, random values are assigned to the variables and passed to the function as input parameters.

```
dcltrans
  transaction TMain
  var
    n1, n2, n3, n4, n5, n6, n7, n8, n9, n10 : number;
    f1, f2, f3, f4                          : float;
    s1, sRet                                : string;
  begin
    n1   := 127; n2 := 32000; n3 := 2000000;
    n4   := 64000; n5 := 255;
    f1   := 12345.12345; f2 := 12.99;
    s1   := "Teststring";
    sRet := t_param(n1, n2, n3, f1, f2, n4, n5, s1,
                    n6, n7, n8, f3, f4, n9, n10);
  end TMain;
```

Structured Data Types in External Functions

When using structured data types, it is important to pay attention to the alignment and size of structure elements. The table below shows the size of all conventional data types that can be used within function declarations in Silk Performer scripts.

Simple data types allowed as input and output parameters are:

Table 3: Conventional data type sizes

Data Type	Size
Char, unsigned char	1
Short, unsigned short	2
Long	4
Double	8
Float	4
Char[x]	x

C Source Code

The example below clarifies structure composition by showing a data structure that contains all the simple data types.

```
// declare a demo data structure
#pragma pack(1) // alignment to 1-byte boundaries
typedef struct {
  char           myChar;       // size: 1 byte pos: 1
  short          myShort;      // size: 2 bytes pos: 2
  long           myLong;       // size: 4 bytes pos: 4
  double         myDouble;     // size: 8 bytes pos: 8
  float          myFloat;      // size: 4 bytes pos: 16
  unsigned short myUShort;     // size: 2 bytes pos: 20
  char           myString[10]; // size: 10 bytes pos: 22
  unsigned char  myUChar;      // size: 1 byte pos: 32
} TESTSTRUCT;
#pragma pack()

 char* T_STRUCT(TESTSTRUCT * s)
{
  char sBuf[10000];
  // manipulate values of data structure and generate a
  // return value for demonstration purpose
  ...
  return sBuf;
}
```

BDL External Function Declarations

To handle structured data types, BDL uses strings of appropriate length and number of type-dependent `Get` and `Set` functions. For each structured data type passed to a function, an inout string parameter must be defined. These strings must be exactly as long as the data structures defined in the DLLs.

The first position in a BDL string that can hold a structured data type is one (1). When variables are aligned to one-byte boundaries, the starting position of the next element is derived by adding the current position to the size of the current element. For the data structure declared in the example above, a 32-byte string is required.

```
"T_STRUCT"
  function t_struct(inout string(32)): string(1000);
```

BDL External Function Calls

In the following example, arbitrary values are assigned to all the elements of the data structure using the corresponding `Set` functions. Next, the external function `t_struct` is called to manipulate the data structure.

```
dcltrans
  transaction TMain
  var
    n1, n2, n3, n4, n5  : number;
    n6, n7, n8, n9, n10 : number;
    f1, f2, f3, f4      : float;
    sStruct             : string(100);
    sRet                : string(1000);
  begin
    /* one-byte alignment */
    SetChar (sStruct, 1, 127);
    SetShort (sStruct, 2, 32000);
    SetLong (sStruct, 4, 2000000);
    SetDouble(sStruct, 8, 12345.12345);
    SetFloat (sStruct, 16, 12.99);
    SetUShort(sStruct, 20, 64000);
    SetString(sStruct, 22, "123456789", 10);
    SetUChar (sStruct, 32, 255);
    sRet := t_struct(sStruct);
  end TMain;
```

Client IP Address Simulation

This chapter explores the client IP address simulation feature of Silk Performer. The method of configuring network interface cards for multiple IP addresses is explained, as are the script functions used to set up local IP addresses. A brief introduction describes how to activate newly installed/assigned IP addresses.

Multiple IP Addresses per NIC

If a computer is configured with more than one IP address, it is referred to as a multi-homed system. Multi-homed systems are supported in one of three ways:

- Multiple IP addresses per NIC (logical multi-homed): Using the Control Panel, five addresses per card may be configured. However, more addresses may be added to the registry.
  - Requirements: NT4, SP4
  - The total number of IP addresses allowed per NIC depends on the installed NIC. Based on the test results, a 3COM905BTX can handle about 2000 IP addresses.
- Multiple NICs per physical network (physical multi-homed):
  - No restrictions other than hardware.
- Multiple networks and media types:
  - No restrictions other than hardware and media support.

Adding IP Addresses in the System Configuration Wizard

1. Disable DHCP and add gateway and DNS addresses.
2. Select **Tools** > **System Configuration Manager**.
3. Connect to the machine to which you want to add new IP addresses: Click **Connect to**, select a machine and click **Connect**.

   Note: By default, you will be connected to the localhost. If you cannot connect to a particular host, make sure that the Silk Launcher Service is running on that host.
4. Click the **IP Address Manager** tab and select the network adapter to which you want to add new IP addresses and click **Add**.
5. Specify the IP addresses you want to add.
   - **Number**: The number of sequential IP addresses to be added subsequent to the one specified in the **From IP address** field.
   - **To IP address**: Any number of IP addresses will be added until this IP is matched. Addresses must be sequential and can either increase or decrease.

   Note: If you use the **From IP address** and the **To IP address** fields, make sure that the **To IP address** value is higher than the **From IP address** value. For example: From: 192.12.23.1, To: 192.12.23.108
6. Reboot the machine.
7. To verify that the IP addresses you added are valid, either:

- Call `ipconfig.exe` (within the shell) to verify that all IP addresses are configured properly.
- Launch the **System Configuration Manager** to test that the IP addresses were added correctly.

Note: By clicking the **Check** button with the Host name or IP address field blank, the System configuration Manger checks that all added IP addresses are working correctly. If you enter a particular IP address into the Host name or IP address field, only that address will be checked.

During replay you can use a third-party tool to examine the bindings to the local virtual IP address. TCPView can be downloaded from *http://www.sysinternals.com*.

Setting Dynamic User IP Addresses Using Automatic Round-Robin

1. Select **Settings** > **Active Profile**.
2. On the left side of the **Profile** dialog box, click **Internet**.
3. Check the **Client IP address multiplexing** check box.
4. Run your test and use a utility such as TCPView to verify your IP address multiplexing.

Setting Dynamic User IP Addresses Using Script Functions

To set up dynamic user IP addresses with functions, use the `WebSetLocalAddress` and the `WebTcpipConnect` functions.

For sample applications, review the file `WebMultipleClientIpAddresses01.bdf`.

Routing Problems Due to Client IP Address Simulation

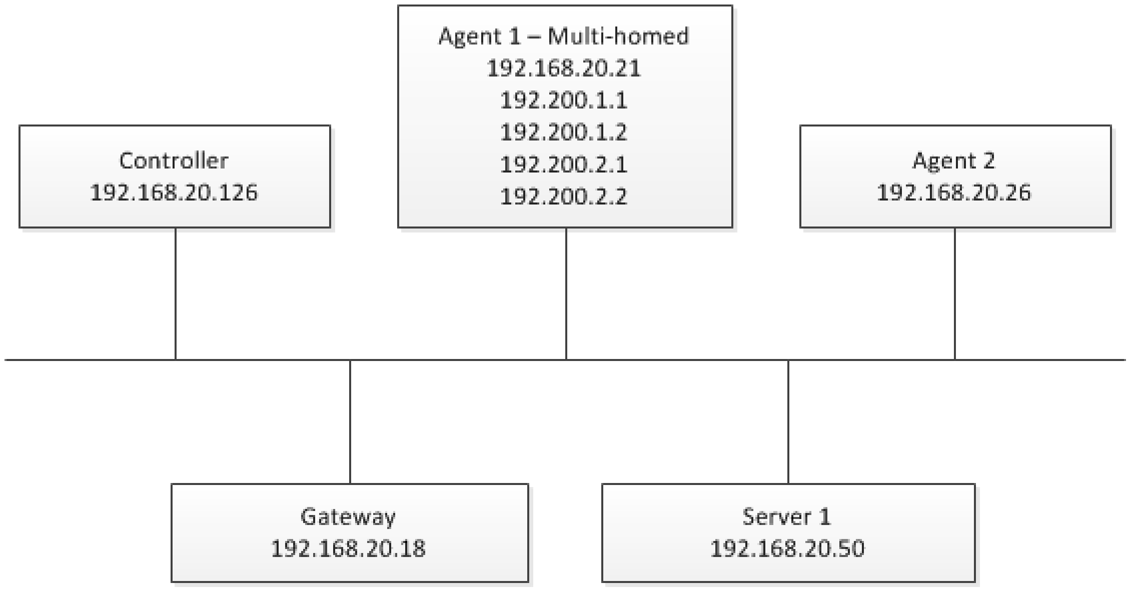
- **Sending packets from multi-homed clients to servers:** With multi-homed clients, the destination IP address is always the IP address of the selected server. There are no differences between network operations, and problems usually do not arise when delivering packets from multi-homed clients.
- **Sending packets from servers to multi-homed clients:** Problems arise when a server attempts to send back a reply to a multi-homed client using the destination IP address of the sending client application. If the IP address is a local subnet-address, the packet will find its way back to the client because there is already a correct entry in the server's routing table. If the selected IP address does not belong to the local subnet and the server does not find another matching entry in its routing table (this is the normal behavior when entries have not been added) it sends the packet to the default gateway (using the default entry). If it does not find a default entry, it generates a network unreachable error.

Solutions for Routing Problems Due to Client IP Address Simulation

There are several options for configuring servers to send responses to multi-homed clients. Here are two options for different network configurations:

Server and agent on the same subnet

If there is no router between the agent and the server, you must add entries to the routing table of the server. If the generated IP addresses of the client begin with the same two numbers (for example 192.200.), you only need to add one entry to the routing table (`route add 192.200.0.0 mask 255.255.0.0<your normal IP address>`). The server will consider the client as the appropriate router for all addresses beginning with 192.200.



In the example below, server 1 receives a packet from agent 1 and attempts to send its answer from 192.168.20.50 to 192.200.2.1. If you do not modify the server configuration (which means the server does not know that agent 1 is multi-homed), server 1 will send the packet to the default gateway, because there is only one matching entry in the routing table.

Network destination	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	192.168.20.18	192.168.20.126	1

If you call (at server 1) `route add 192.200.0.0 mask 255.255.0.0 192.168.20.21`, a new entry will be added. The entry resembles the following:

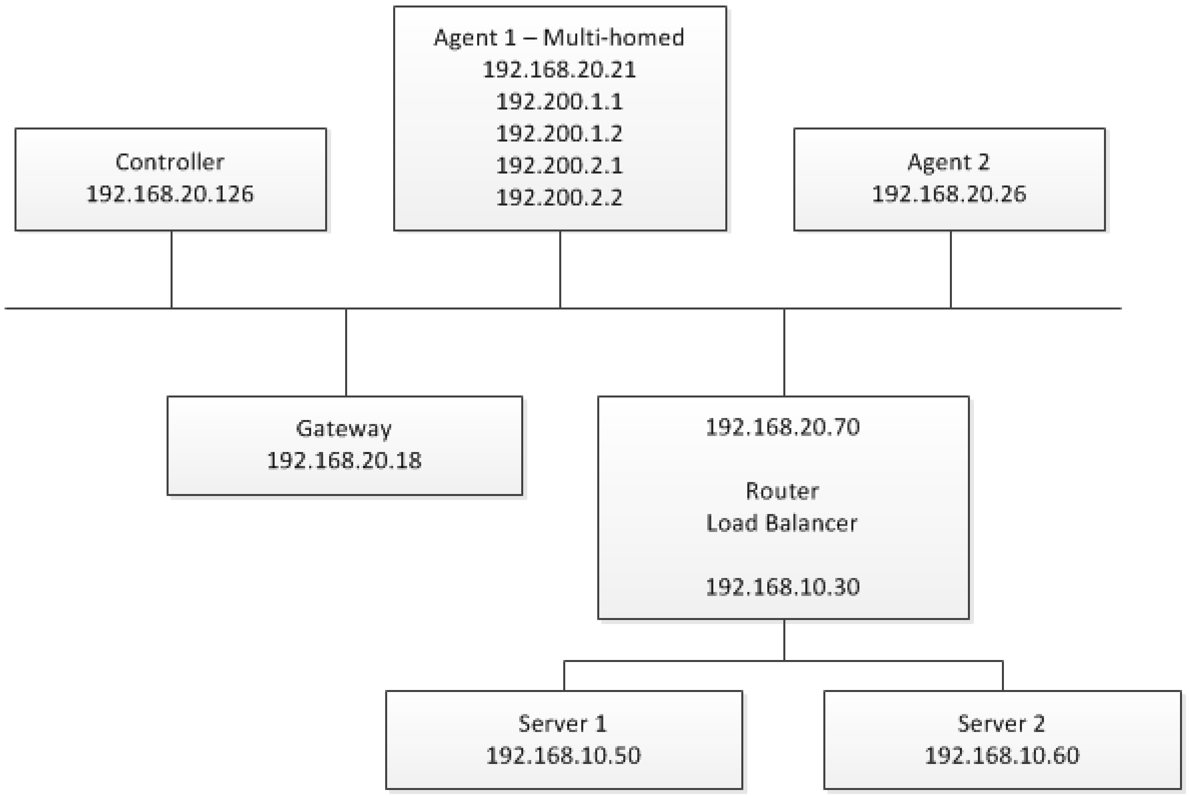
Network destination	Netmask	Gateway	Interface	Metric
192.0.0.0	255.255.0.0	192.168.20.21	192.168.20.50	1

As this entry has a higher priority than the entry of the default gateway, server 1 will send all packets with a destination address type of 192.200.x.x to agent 1 (believing this to be the correct gateway).

Router or load-balancer between the server and the agent

A router or load-balancer is positioned between the server and the agent. In such a case, you must alter the routing table of the router. If agent 1 sends a packet to server 1 (from 192.200.2.1 to 192.168.10.50) and agent 1 is configured correctly, that means it has a routing table entry that resembles the following:

Network destination	Netmask	Gateway	Interface	Metric
192.168.1.0	255.255.0.	192.168.20.70	192.168.20.21	1



Agent 1 will send the packet to the router, which will in turn forward the packet to server 1. Now server 1 wants to send a response to the request of agent 1 (from 192.168.10.50 to 192.200.2.1). Because of its default entry in the routing table, server 1 sends the packet to the router. However, now problems arise because the router knows nothing of the new IP addresses of agent 1 and will use its default route (shown below) and send the packet to the gateway.

Network destination	Netmask	Gateway	Interface	Metric
0.0.0.0	0.0.0.0	192.168.20.18	192.168.20.70	1

If you call (at the router) `route add 192.200.0.0 mask 255.255.0.0 192.168.20.21`, a new entry will be added, the resembles the following:

Network destination	Netmask	Gateway	Interface	Metric
192.200.0.0	255.255.0.0	192.168.20.21	192.168.20.70	1

Because this entry has a higher priority than the entry of the default gateway, the router will send all packets with a destination address type of 192.200.x.x to agent 1 (believing it to be the correct gateway).

Note: You can configure the default gateway of your subnet to forward all packets from the server to the multi-homed agent. However, this might cause problems when a router is forced to send out a packet through the same interface by which the packet is received, the router thinks that the server that originally sent the packet made an incorrect routing decision. The router then generates an ICMP redirect error. This also increases load on the network and server.

Testing Routing Adaptations

Use the **System Configuration Manager** to test if newly added IP addresses and routing adaptations work correctly:

1. Select **Tools** > **System Configuration Manager**. The **System Configuration Manager** window appears.

2. Connect to the machine to which you added new IP addresses.

   Note: By default, you will be connected to the localhost. If you cannot connect to a particular host, make sure that the Silk Launcher Service is running on the host.
3. Click the **IP Address Manager** tab.
4. Select the network adapter to which you added new IP addresses.
5. In the **Network** section, enter the host name or IP address of the server you want to load test.
6. Click **Check**. If no error window appears, all listed IP addresses will have a route is up mark in the Test result section of the list window.

Defining User Types

The next step in the process of conducting a Silk Performer load test is to define one or more user types for the active workload. A user type is a unique combination of a script, a user group, and a profile. By selecting a user type, you determine which script will be executed with which user group and profile.

To find out what your active workload is, expand the **Workloads** node in the **Project** tree. The active workload is shown in bold text.

Click **Define User Types** on the workflow bar to access all necessary settings.

Note: The **Define User Types** button on the workflow bar is only visible when the simple workflow bar is enabled. The workflow bar displays the simplified workflow by default. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar**, **Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Defining a User Type

To define user types for the active workload:

1. Click **Define User Types** on the workflow bar. The **Workflow - Define User Types** dialog box appears.
2. *Optional:* Narrow down the list of **Available User Types** by selecting entries from the **Script** and **Profile** lists.
3. *Optional:* Click **Add new** to create a new profile. If you select a profile from the list, you can click **Edit Profile** to adjust the settings of the profile. For example, you can define a certain browser and bandwidth for this profile. Click **Set as active** to make the selected profile the active one.
4. Select one or more user types in the **Available User Types** list and click the arrow buttons to assign the user types to the workload. You can also double-click user types to move them between the **Available User Types** list and the **User Types in Workload** list.
5. User types that are assigned to the workload are automatically selected for execution. If you want a certain user type to not be executed during the baseline run, deselect it.
6. Click **Next** to advance to the next step in the workflow.

Note: The **Define User Type** button is only visible when simple workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

User Types

A *User type* is a unique combination of a script, a user group, and a profile.

Silk Performer automatically generates all possible combinations of user types out of the available scripts, user groups and profiles. To have all user types that you need available, you must add the desired profiles to your test project beforehand. For example, to emulate three different bandwidths during your load test, you must first add a profile for each bandwidth.

The **Find Baseline** or the **Define User Types** dialog box allow you to assign user types to the active workload.

You can run a baseline test to establish the baseline performance of your load test for the selected user types. The different user types can be run at the same time.

Only one virtual user per user type is executed for the baseline test. The think times contained in the script are not randomized for comparison reasons.

Finding Baselines

A baseline serves as a reference level for subsequent load tests. Silk Performer uses the results of a baseline test to calculate the number of concurrent users per user type. Additionally, you can set response time thresholds based on the results of a baseline.

At first, you need to run a baseline test. After a baseline test run is finished, you can view all measurements and details of the run on the **Baseline Test Summary** page and in the baseline report. If you are happy with the results, you can set the baseline test as your new baseline. You can view the results and values of your baseline at any time by clicking **View Baseline** on the workflow bar. This opens the **Baseline Summary** page.

Note: To work with baselines, you must enable the full workflow bar.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Finding a Baseline

Customize your test script by assigning a user profile to it before running a baseline test.

1. Click **Find Baseline** on the workflow bar. The **Workflow - Find Baseline** dialog box appears.
2. *Optional:* Narrow down the list of **Available User Types** by selecting entries from the **Script** and **Profile** lists.
3. *Optional:* Click **Add new** to create a new profile. If you select a profile from the list, you can click **Edit Profile** to adjust the settings of the profile. For example, you can define a certain browser and bandwidth for this profile. Click **Set as active** to make the selected profile the active one.
4. Select one or more user types in the **Available User Types** list and click the arrow buttons to assign the user types to the workload. You can also double-click user types to move them between the **Available User Types** list and the **User Types in Workload** list.
5. User types that are assigned to the workload are automatically selected for execution. If you want a certain user type to not be executed during the baseline run, deselect it.
6. Click **Run** to perform the baseline test.

Silk Performer runs a baseline test to calculate average measures against which future test runs are measured.

Note: The **Find Baseline** button is only visible when the full workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Baseline Tests

A baseline test enables you to determine your application's ideal performance baseline.

With a baseline test, a customized test is run with just one user per user type. And the results from this unstressed performance of the application form the basis for calculating the number of concurrent users per user type and for setting the appropriate boundaries for the HTML page response times and transaction response times. Additionally, the required bandwidth for running the load test is calculated from the baseline results. The measurements typical of a real load test are used, and report and output files are generated.

Additionally, a baseline test can serve as a trial run of your test script. A trial run can help you verify the following:

- Customizations have not introduced new errors into the script.
- The script can accurately and fully reproduce the interaction between the client application and the server.

Note: Baseline tests ignore the **Random thinking time** option. This makes it easier for you to compare the results of different baseline tests because the think times remain constant between tests.

For baseline test runs, the following settings are configured automatically:

- A **Baseline report file** is automatically created.
- The **Stop virtual users after simulation time (Queuing Workload)** option is enabled.
- The **Random thinking time** option is disabled.
- The **Loadtest description** text box is set to `Baseline Test`.
- The **Display All Errors Of All Users** option in the Monitor window is enabled.
- The **Virtual user output files (.wrt)** option is enabled.
- The **Virtual user report files (.rpt)** option is enabled.

Baseline Test Summary

When a baseline test run is complete, the **Baseline Test Summary** page appears. You can also open this page from the **Results** tree. You can perform the following actions in the **Next Steps** area on the right side:

- Click **View baseline test report** to view all the metrics of the baseline test run in detail.
- Click **Set as baseline** to make the test you have just run your baseline (your reference level) for the upcoming load tests.
- Click **Compare with baseline** to compare the results of the test you have just executed with the current baseline. This button is only visible if there is already a baseline set.
- Click **Adjust Workload** to perform the next step in the workflow. This button is only visible if you have not yet defined a workload.

You can always return to the **Baseline Test Summary** page by opening it from the **Results** tree.

Note: When you click **Set as baseline**, the **Baseline Summary** page appears. Do not confuse the **Baseline Test Summary** page with the **Baseline Summary** page. The **Baseline Test Summary** page appears when a baseline test is complete. The **Baseline Summary** page appears when you have set the baseline test as your baseline or when you click **View Baseline** on the workflow bar.

You can run one or more user types in a baseline test. If you set the results as baseline, the user types and their results will be added to the baseline. If a baseline with results for a particular user type already exists, these results will be overwritten.

Note: If you want to prevent the summary page to appear each time a test is complete, disable the **Show Summary Page** button in the toolbar of the **Monitor** page.

User Types

A *User type* is a unique combination of a script, a user group, and a profile.

Silk Performer automatically generates all possible combinations of user types out of the available scripts, user groups and profiles. To have all user types that you need available, you must add the desired profiles to your test project beforehand. For example, to emulate three different bandwidths during your load test, you must first add a profile for each bandwidth.

The **Find Baseline** or the **Define User Types** dialog box allow you to assign user types to the active workload.

You can run a baseline test to establish the baseline performance of your load test for the selected user types. The different user types can be run at the same time.

Only one virtual user per user type is executed for the baseline test. The think times contained in the script are not randomized for comparison reasons.

Baseline Test Report

After you have run a baseline test, the **Baseline Test Summary** page appears. Click **View baseline test report** to view a detailed report with all results and metrics of the baseline test. The detailed test report can be displayed for any test except a Try Script run.

The detailed test report is an XML/XSL-based report that provides you with a summary table, transaction response-time details, timers for all accessed HTML pages, Web forms, and errors that occurred. This information includes the most important test results in a tabular form and is available for all user types involved in the test.

Detailed Test Report Content

A detailed test report consists of the following sections:

- General information
- User types
    - Summary tables
    - Transaction response times
    - HTML page and action timers
    - Web form measurements
    - Accept Results button

General Information

The **General Information** section includes administrative information in tabular form.

Administrative information includes the Silk Performer Version information, the project name, a description of the project, the date and time of the baseline test, the workload definition, the workload model, and the number of errors that occurred.

User Types

For each user type involved in the test run, a separate section is available with details on the measured response times. The following information appears in the summary line:

- Number of virtual users (one for a baseline test)
- Test duration
- Session time
- Session busy time
- Average page time
- Average action time
- The following numbers:
    - Transactions that were executed successfully
    - Canceled transactions
    - Failed transactions
    - Errors

The session time consists of the execution time of all transactions defined in the `dcluser` section of a test script, without the initial and end transactions. The session busy time is calculated as the session time minus think times, as the following example shows.

```
dcluser
  user
    Vuser1
  transactions
    TInit  : begin;
    T1     : 1;
    T2     : 3;
    Tend   : end;
```

The session time is the average response time of T1 + 3 * average response time of T2. The session busy time is the same without any think time.

If you are satisfied with the results, you can set them as baseline results and save them for further processing, such as the calculation of the number of concurrent virtual users and the network bandwidth needed for the load test.

For each user type, detailed results are available in the following sections:

- **Summary tables** – Contains aggregate measurements of all virtual users in tabular form. The first table provides general information, such as the number of transactions that were executed and the number of errors that occurred. The remaining tables provide summary information that is relevant to the tested application type.
- **Transactions** – Displays the following aggregate measurements in tabular form for all transactions of a specific user type:
    - Transaction response time – Measured from the beginning to the end of the transaction, including all think times.
    - Transaction busy time – The transaction response time without any think time.
- **Page and action timers** – Displays the following summary measurements in tabular form for every accessed Web page:
    - Download time of the entire page
    - HTML-document download time
    - Server busy time
    - Amount of data downloaded with the page
    - Amount of data downloaded for embedded objects
- **Web forms** – Displays the following measurements in tabular form for every used Web form:
    - Roundtrip time
    - Server busy time
    - Number of HTTP hits
    - Amount of request data that was sent
    - Amount of response data that was received

If the results of an inspected load test are acceptable for use as a baseline, you can store them for further calculations and processing by clicking **Accept Results**. The results of tests with all user types involved are stored in the file `<projectdir>\BaselineResults\baselineReport_<workloadname>.brp`. You must define a separate baseline for every workload that is defined in your load test project. If you create a copy of a workload, the baseline results are also copied.

Setting a Baseline Test as Baseline

Before you can set a baseline, you must run a baseline test:

1. Click **Find Baseline** on the workflow bar. The **Workflow - Find Baseline** dialog box appears.
2. Click **Run** to start the baseline test.

For more information on this dialog box, see *Finding a Baseline*.

3. When the baseline test run is finished, the **Baseline Test Summary** page appears.
4. Click **Set as baseline** in the **Next Steps** area on the right side.
5. Click **Yes** on the confirmation dialog box.

The baseline test is set as baseline and the **Baseline Summary** page appears.

Note: The buttons **Find Baseline** and **View Baseline** are only visible if the full workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Viewing Baselines

The next step in the process of conducting a Silk Performer load test is to set the baseline test (see the previous step *Finding Baselines*) as your baseline. The baseline will serve as a reference level for subsequent load tests and should reflect the desired performance of the application under test. Once you have set a baseline test as your baseline, you can configure response time thresholds.

Note: To work with baselines, you must enable the full workflow bar.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Baseline Summary

When you have clicked **Set as baseline** on the **Baseline Test Summary** page, the **Baseline Summary** page appears. You can perform the following actions in the **Next Steps** area on the right side:

- Click **View baseline report** to view all the metrics of the baseline in detail.
- Click **Set response time thresholds** to configure in what timeframe the subsequent load tests should respond.

You can always return to the **Baseline Summary** page by clicking **View Baseline** in the workflow bar.

Note: Do not confuse the **Baseline Test Summary** page with the **Baseline Summary** page. The **Baseline Test Summary** page appears after you have run a baseline test. The **Baseline Summary** page appears when you set the baseline test as your baseline or when you click **View Baseline** on the workflow bar.

Response Time Thresholds

With the accepted results from the previous baseline tests, thresholds for response times can be set for subsequent load tests. These thresholds are used in the general overview report of a load test to separate good, acceptable, and unacceptable Web response times.

Set boundaries to identify the following types of response times:

- *Good* response times fall below boundary 1.
- *Unacceptable* response times fall above boundary 2.
- *Acceptable* response times fall between these boundary values.

Set thresholds for the following objects:

- Transaction response times
- Custom timers

- HTML page timers

Specify a multiplier for the calculation of the boundaries from the baseline results. The average response times of the timers are multiplied by these factors, and the boundaries are set accordingly. For example, a multiplier of 3 means that you set a boundary three times higher than the average response time of the timer in the baseline test.

If the measured value in the baseline test is `0`, you can specify a minimum value for the boundaries. The load testing scripts connected with the user types is updated to set the thresholds accordingly.

Setting Response Time Thresholds

1. Click **View Baseline** on the workflow bar. The **Baseline Summary** page appears.
2. Click **Set response time thresholds** in the **Next Steps** area on the right side. The **Automatic Threshold Generation** dialog box opens.
3. Select the timers for which you want to set thresholds.
4. Specify the appropriate **Lower bound** and **Upper bound** multipliers for your load tests.
5. If the corresponding timers from the baseline test are `0`, specify minimum values.
6. If you want to raise an error or a warning message in case a threshold is exceeded, you can specify the severity of the raised message.
7. Click **OK**.

Note: The buttons **Find Baseline** and **View Baseline** are only visible when the full workflow bar is enabled.

Note: By default, Silk Performer displays the simple workflow bar. To switch between the workflow bar types, right-click on the workflow bar and click **Show Simple Workflow Bar, Show Full Workflow Bar**, or **Show Monitoring Workflow Bar**.

Performance Levels

When a load test is completed, a performance engineer usually has to assess whether the test has been successful or not. This assessment is typically based on a thorough analysis of the load test results.

Performance engineers can use the broad range of analysis tools that the Workbench and Performance Explorer provide. They can use the various reports to analyze a number of metrics, such as the amount of errors or the response time thresholds. They could also compare response times to performance criteria or service levels, which are usually determined prior to the load test execution.

This process can be described as manual load test assessment. However, in many cases an automatic assessment of the load test is useful. For example: In continuous integration environments, users typically want to be notified only in case performance criteria have been violated. In Silk Performer, you can use the so-called *performance levels* for such scenarios. Performance levels can be defined in two ways: proactively and reactively.

- **Proactive approach**: A performance engineer runs a baseline test, then defines performance levels (based on the baseline test results), and assigns the performance levels to the key measures of a test. Now the engineer executes the load test. When the execution is completed, the engineer opens an overview report and checks which performance levels have been missed.
- **Reactive approach**: The performance engineer executes the load test first and, only then, defines the performance levels. The load test results are used as a guideline for defining the performance levels.

Of course, these approaches can also be combined. This might be necessary, if performance levels have been defined prior to a load test execution and need to be adapted when the execution is completed, because the reports do not reflect the desired status.

To cover both approaches, you can access the performance level dialogs through the Workbench and Performance Explorer. In the Workbench, click **Project** > **Define Performance Levels** or **Project** >

Assign Performance Levels. In Performance Explorer, click the **Reports** tab and click **Define Levels** or **Assign Levels**.

Defining and Assigning Performance Levels (Proactive Approach)

To assign performance levels to measures, Silk Performer needs to know the measures that will be available after a load test. Therefore, make sure to run a baseline test and set a baseline before you follow the steps described below. If no baseline is set, there are no measures available to assign performance levels to.

1. In the Workbench, click **Project** > **Define Performance Levels**.
2. The **Define Performance Levels** dialog displays, containing a table. The table rows represent the performance levels. You can add as many performance levels as you require.
3. The columns represent the performance indicators. Per column, select one of the following performance indicators:
   - **Avg**: average value
   - **P50**: median
   - **P90**: 90th percentile
   - **P95**: 95th percentile
   - **P99**: 99th percentile
4. Name your performance levels and enter values for each level.
5. When your performance levels are defined, click **Assign Levels**.
6. The **Assign Performance Levels** dialog shows all measures that were collected during the baseline test. You can assign the defined performance levels manually to each measure. Or you can click **Automatically set levels based on baseline**. You can also filter the list to quickly find a specific measure, and you can click **Reset all levels to none**. When you are done, click **OK**.

Complete modeling your load test and execute it. When the execution is completed, open an overview report and look out for the performance levels. They are color-coded: Performance levels that are met display in green, performance levels that are missed display in red. This allows you to instantly see whether a load test is to be considered as passed or failed. To tweak your settings after the load test execution, follow the reactive approach.

Note: A performance level definition is valid for a particular workload. If you have multiple workloads, you can define performance levels for each workload.

Defining and Assigning Performance Levels (Reactive Approach)

1. Model and execute a load test.
2. When the execution is completed, the **Load Test Summary** page displays. Click **Analyze load test**. Performance Explorer opens and the HTML Overview Report displays.
3. Click the **Reports** tab and **Define Levels**.
4. The **Define Performance Levels** dialog displays, containing a table. The table rows represent the performance levels. You can add as many performance levels as you require.
5. The columns represent the performance indicators. Per column, select one of the following performance indicators:
   - **Avg**: average value
   - **P50**: median
   - **P90**: 90th percentile
   - **P95**: 95th percentile
   - **P99**: 99th percentile
6. Name your performance levels and enter values for each level.

7. When your performance levels are defined, click **Assign Levels**.
8. The **Assign Performance Levels** dialog shows all measures that were collected during the load test. You can assign the defined performance levels manually to each measure. Or you can click **Automatically set levels based on baseline**. You can also filter the list to quickly find a specific measure, and you can click **Reset all levels to none**. When you are done, click **OK**.

Look out for the performance levels in the overview report. They are color-coded: Performance levels that are met display in green, performance levels that are missed display in red. This allows you to instantly see whether a load test is to be considered as passed or failed. To set your performance levels before executing the load test, follow the proactive approach.

Note: A performance level definition is valid for a particular workload. If you have multiple workloads, you can define performance levels for each workload.

Adjusting Workload

Configuring workload is part of the process of conducting a load test. Silk Performer offers different workload models to be used as a basis for your load test. Before configuring workload, you must select the model that best fits your needs.

You can define more than one workload model in your load test project and save them for further usage, but only one workload model can be active at a time. The accepted baseline results are associated with a workload model. If you copy or rename a workload model, the accepted baseline results are copied or renamed accordingly.

Workload Models

Silk Performer provides the following workload models:

- **Increasing** – At the beginning of a load test, Silk Performer does not simulate the total number of users defined. Instead, it simulates only a specified part of them. Step by step, the workload increases until all the users specified in the user list are running.

  This workload model is especially useful when you want to find out at which load level your system crashes or does not respond within acceptable response times or error thresholds.
- **Steady State** – In this model, the same number of virtual users is employed throughout the test. Every virtual user executes the transactions defined in the load-testing script. When work is finished, the virtual user starts again with executing the transactions. No delay occurs between transactions, and the test completes when the specified simulation time is reached.

  This workload model is especially useful when you want to find out about the behavior of your tested system at a specific load level.
- **Dynamic** – You can manually change the number of virtual users in the test while it runs. After the maximum number of virtual users is set, the number can be increased or decreased within this limit at any time during the test. No simulation time is specified. You must finish the test manually.

  This workload model is especially useful when you want to experiment with different load levels and to have the control over the load level during a load test.
- **All Day** – This workload model allows you to define the distribution of your load in a flexible manner. You can assign different numbers of virtual users to any interval of the load test, and each user type can use a different load distribution. Therefore, you can design complex workload scenarios, such as workday workloads and weekly workloads. You can also adjust the load level during a load test for intervals that have not started executing.

  This workload model is especially useful when you want to model complex, long lasting workload scenarios in the most realistic way possible.
- **Queuing** – In this model, transactions are scheduled by following a prescribed arrival rate. This rate is a random value based on an average interval that is calculated from the simulation time and the number of transactions per user specified in `dcluser` section of your script. The load test finishes when all of the virtual users have completed their prescribed tasks.

Note: With this model, tests may take longer than the specified simulation time because of the randomized arrival rates. For example, if you specify a simulation time of 3,000 seconds and want to execute 100 transactions, then you observe an average transaction arrival rate of 30 seconds.

This workload model is especially useful when you want to simulate workloads that use queuing mechanisms to handle multiple concurrent requests. Typically, application servers like servlet engines or transaction servers, which are receiving their requests from Web servers and not from end users, can be accurately tested by using the queuing model.

- **Verification** – A verification test run is especially useful when combined with the extended verification functionality. This combination can then be used for regression tests of Web-based applications. A verification test performs a specified number of runs for a specific user type.

  This workload is especially useful when you want to automate the verification of Web applications and when you want to start the verification test from the command line interface.

Configuring Increasing Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select the workload model **Increasing**.
3. Select a user type and set the following values:
   a) **Ramp-up time**: Specify the initial number of virtual users (**Start VUsers**), the maximum number of virtual users (**max. VUsers**), and the number of virtual users to be added in a specific time interval (**Add x per x**). Based on these values, Silk Performer calculates the ramp-up time.
   b) **Steady time**: Specify the steady time.
   c) **Ramp-down time**: Specify the number of virtual users to be stopped in a specific time interval (**Per x stop x**).
   d) **Warm-up time**: During the warm-up time, Silk Performer does not measure the performance. The measuring starts after the warm-up time. Enter zero to start the measuring immediately. You can use the warm-up time to prepare the application under test for the actual load test. If you do not want to take measures during this preparation phase, you can use the warm-up time.
   e) **Measurement time**: Specify a measurement time. If the measurement time is set to `0`, all measures taken between the warm-up time and the end of the simulation time are considered in the reports.
   f) **Close-down time**: When the close-down time starts, Silk Performer will stop measuring the performance. To stop the load test right after the measurement time, make sure that your warm-up and measurement times sum up to the simulation time. You can use the close-down time to shut down the application under test or to close services if you do not want to take measures during this tidying-up phase.

   Note: The sum of the **Ramp-up time** and the **Steady time** must not be shorter than the sum of the **Warm-up time** and the **Measurement time**.

   Tip: You can enter the time in the format `hh:mm:ss` or you can enter for example `1h12m30s`. Silk Performer will automatically convert `1h12m30s` into `01:12:30`. If you just enter `7m`, it will be converted into `00:07:00`.
4. Click **Next**. The **Workflow - Assign Agents** dialog box appears.

Configuring Steady State Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select the workload model **Steady State**.
3. Select a user type and set the following values:
   a) **max. VUsers**: The number of virtual users that will be run during the load test.
   b) **Steady time**: Defines how long the load test will last.

c) **Ramp-down time**: Specify the number of virtual users to be stopped in a specific time interval (**Per x stop x**).
d) **Warm-up time**: During the warm-up time, Silk Performer does not measure the performance. The measuring starts after the warm-up time. Enter zero to start the measuring immediately. You can use the warm-up time to prepare the application under test for the actual load test. If you do not want to take measures during this preparation phase, you can use the warm-up time.
e) **Measurement time**: Specify a measurement time. If the measurement time is set to `0`, all measures taken between the warm-up time and the end of the simulation time are considered in the reports.
f) **Close-down time**: When the close-down time starts, Silk Performer will stop measuring the performance. To stop the load test right after the measurement time, make sure that your warm-up and measurement times sum up to the simulation time. You can use the close-down time to shut down the application under test or to close services if you do not want to take measures during this tidying-up phase.

Tip: You can enter the time in the format `hh:mm:ss` or you can enter for example `1h12m30s`. Silk Performer will automatically convert `1h12m30s` into `01:12:30`. If you just enter `7m`, it will be converted into `00:07:00`.

4. Click **Next**. The **Workflow - Assign Agents** dialog box appears.

Configuring Dynamic Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select the workload model **Dynamic**.
3. Select a user type and set the **max. VUsers** value. This is the maximum number of virtual users that can be run during the load test. It is the upper boundary for upcoming virtual user settings.
4. Click **Next**. The **Workflow - Assign Agents** dialog box appears.

Configuring All Day Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select the workload model **All Day**.
3. Select a user type and set the following values:
   a) **max. VUsers**: The maximum number of virtual users that can be run during the load test. This is the upper boundary for upcoming virtual user settings.
   b) **Simulation time**: Defines how long the load test will last.

   Tip: You can enter the time in the format `hh:mm:ss` or you can enter for example `1h12m30s`. Silk Performer will automatically convert `1h12m30s` into `01:12:30`. If you just enter `7m`, it will be converted into `00:07:00`.
4. Click **Next**. The **All Day Workload Configuration** dialog box appears.
5. Select a user type from the list on the left side and configure intervals by entering values into the grid on the right side. Define a start and an end number of virtual users, as well as the duration for each interval. The chart on the top visualizes your settings.

   Note: When a load test was started, you can still change the number of virtual users for intervals that have not yet started, but you cannot exceed the maximum number of virtual users specified for the load test.
6. Click **OK**. The **Workflow - Assign Agents** dialog box appears.

Configuring Queuing Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.

2. Select the workload model **Queuing**.
3. Select a user type and set the following values:
   a) **max. VUsers**: The number of virtual users that will be run during the load test.
   b) **Steady time**: Specify the steady time.
   c) **Ramp-down time**: Specify the number of virtual users to be stopped in a specific time interval (**Per x stop x**).
   d) **Warm-up time**: During the warm-up time, Silk Performer does not measure the performance. The measuring starts after the warm-up time. Enter zero to start the measuring immediately. You can use the warm-up time to prepare the application under test for the actual load test. If you do not want to take measures during this preparation phase, you can use the warm-up time.
   e) **Measurement time**: Specify a measurement time. If the measurement time is set to `0`, all measures taken between the warm-up time and the end of the simulation time are considered in the reports.
   f) **Close-down time**: When the close-down time starts, Silk Performer will stop measuring the performance. To stop the load test right after the measurement time, make sure that your warm-up and measurement times sum up to the simulation time. You can use the close-down time to shut down the application under test or to close services if you do not want to take measures during this tidying-up phase.

   Tip: You can enter the time in the format `hh:mm:ss` or you can enter for example `1h12m30s`. Silk Performer will automatically convert `1h12m30s` into `01:12:30`. If you just enter `7m`, it will be converted into `00:07:00`.
4. Click **Next**. The **Workflow - Assign Agents** dialog box appears.

Configuring Verification Workload

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select the workload model **Verification**.
3. Select a user type and click **Next**. The **Verification Workload Configuration** dialog box appears.
4. Select a **Profile** and a **Script** from the lists.
5. Select a **Usergroup**.
6. Select an **Agent** from the list.
7. In the **TrueLog** section, click one of the following options:
   - Click **Generate TrueLog** to generate a Truelog file for the virtual user group that you are testing.
   - Click **Generate TrueLog On Error** to generate a TrueLog file only when an error occurs.
8. *Optional:* Check the **Animated** check box to launch the TrueLog Explorer automatically when you start a test.

   Note: You must click **Generate TrueLog** to enable the **Animated** setting.
9. Check the **Stress test** check box to disable the use of wait periods that are specified in Web functions or are invoked by calling the think-time function.

   Neither the wait statement nor the transaction scheduling is affected. This feature is equivalent to running a load test with no delays for human interaction.
10. Click one of the following buttons:
    - Click **Connect** to initialize the connection to the agents without actually starting the load test. On the **Monitor** page, you can then click **Start All** in the toolbar.
    - Click **Run** to start the load test.
    - Click **OK** to close the dialog box and save your changes.
    - Click **Cancel** to close the dialog box and abort your changes.

Workload Tab

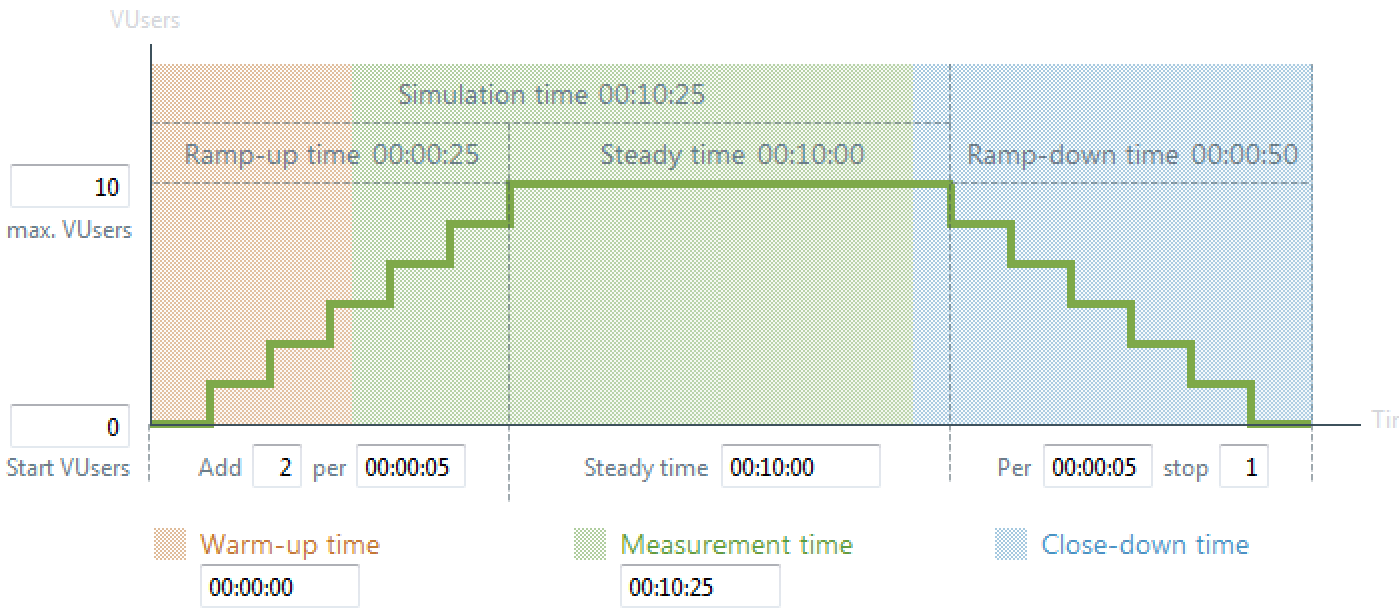
The **Workload Configuration** > **Workload** tab enables you to configure workload-profile settings related to workload model, start time, and monitoring options.

Item	Description
Increasing	Use this option button to switch to the `Increasing` workload model.
Steady State	Use this option button to switch to the `Steady State` workload model.
Dynamic	Use this option button to switch to the `Dynamic` workload model.
All Day	Use this option button to switch to the `All Day` workload model. This selection enables the **Configure All Day Workload** button below.
Queuing	Use this option button to switch to the `Queuing` workload model.
vUsers-over-time graph	The graph at the top of the dialog box graphically represents the selected workload model and uses the data associated with the user group selected in the UserType table.
UserType table	This portion of the dialog enables you to edit the settings of the user types that are associated with your project. Options vary based on the selected workload mode. *(see specific workload-model configuration topics for details on configuring user types.)*
Start time	Set the start time for your test in hours, minutes, and seconds.
Relative	Click this option button to start your test after the specified time period has elapsed.
Absolute	Click this option button to start the test at the specified time.
Automatically start monitoring	Check this check box if you have established a monitoring template and want Performance Explorer to start and stop automatically with the test.
TrueLog On Error	Check this check box to generate TrueLog Explorer files for all transactions that generate errors.
Enable real-time measures	When checked, real-time measure data is generated globally for all virtual users across the entire test. This setting overrides preset profile settings. When unchecked, real-time measure data is disabled for all virtual users globally across the test. This setting overrides preset profile settings. When checked and grayed out, real-time measure data is generated only for those user types who have this setting activated in their corresponding profiles. In this state a tooltip displays information about the user types for which real-time measures have been enabled. **Note:** This check box automatically changes its state when the real-time measure data profile setting option of any virtual user changes.

Item	Description
Loadtest description	*(Optional)* Here you can enter a description of the test for project-management purposes.
Configure All Day Workload	*(enabled only when the All Day workload model is selected)* Opens the **All Day Workload Configuration** dialog.

Time Names in Workload Context

The following time values can be configured in the **Select and adjust Workload** and **Workload Configuration** dialogs:



Measurement	Description
Warm-up time	A load test starts with an optional warm-up time. Measures taken during the warm-up time are not used in the reports. The warm-up time can be used to exclude the start-up tasks of a test (such as allocating and initializing memory or building up caches) from the measured part of the test.
Measurement time	The measurement time is the time between the optional warm-up time and the optional close-down time. During the measurement time, measures are taken and then used in the reports once the test is completed. Typically, the measurement time extends to the end of the simulation time. If the measurement time is set to `0`, all measures taken between the warm-up time and the end of the simulation time are considered in the reports.
Close-down time	A load test ends with an optional close-down time, which is the time between the end of the measurement time and the end of the load test. Measures taken during the close-down time are not used in the reports. The close-down

Measurement	Description
	time can be used to exclude the closing tasks of a test (such as shutting down the virtual users) from the measured part of the test. If the specified simulation time is longer than the sum of the warm-up time and the measurement time, the close-down time starts during the simulation time.

Measurement	Description
Simulation time	The simulation time is the time of the load test excluding the ramp-down time. It can also be described as the sum of the ramp-up time (for increasing workloads) and the steady time.
Ramp-up time	During the ramp-up time, Silk Performer gradually increases the number of virtual users. The ramp-up parameters can be set in the **Select and adjust Workload** dialog or in the **Workload Configuration** dialog. For example, you can specify to start a load test with 10 virtual users (**Start**) and add 2 virtual users (**Add**) every 30 seconds (**Interval**) until reaching a maximum of 20 virtual users (**max. VUsers**). The ramp-up time can only be specified for workloads using the **Increasing** workload model.
Steady time	During the steady time, no more virtual users are added. The number of virtual users remains steady during this time and equals the specified maximum number of virtual users (**max. VUsers**).
Ramp-down-time	During the ramp-down time, Silk Performer gradually decreases the number of virtual users to zero. Since the ramp-down happens during the close-down time, it does not influence the reports. If an end transaction is defined for a virtual user and the profile setting **Call end transaction for stopped virtual users** is enabled, the end transaction will be executed during the ramp-down time. Specifying a ramp-down time also allows you to gradually stop running virtual users to avoid that all virtual users perform the end transaction at the same time. The ramp-down parameters can be set in the **Select and adjust Workload** dialog or in the **Workload Configuration** dialog. For example, you can specify to stop 10 virtual users (**Stop**) every 30 seconds (**Interval**). This translates into a certain ramp-down time depending on the number of virtual users defined in the workload. If the ramp-down time is set to `0` and the profile setting **Call end transaction for stopped virtual users** is enabled, all related virtual users execute the end

Measurement	Description
	transaction simultaneously. If no end transaction exists, the virtual users stop immediately.

Initializing Workload Settings

Silk Performer allows you to specify the same workload settings for multiple user types at a time.

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Click **Initialize**. The **Initialize Workload Settings** dialog appears.
3. Select a **Workload model** from the list.
4. If you want to define one specific value for all user types, use the check boxes: Enable the respective check box and enter the value. To define varying values, enter these directly in the table.
5. Click **OK** to save your changes.

Note: The **Initialize Workload Settings** dialog also allows you to define a duration for the **Ramp-up Time** and **Ramp-down Time**. In contrast, the **Select and adjust workload** dialog allows you to define an increase or decrease value per time. Example for defining the duration: Gradually stop all virtual users within 2 minutes. Example for defining a decrease value: Stop 2 virtual users every 10 seconds.

Tip: The **Initialize Workload Settings** dialog also allows you to copy and paste values from and to spreadsheet programs like Microsoft Excel. To do so, use the keyboard shortcuts **Ctrl+C** and **Ctrl+V**.

Assigning Agents

Explains how to configure and assign individual agents and clusters of agents to your project.

Assigning Agents

In support of large-scale load testing, Silk Performer has consolidated all agent-to-workload assignment features within a single workflow step, available via the **Assign Agents** workflow bar button. Here you can configure the distribution of virtual users in your load testing environment and assign VUsers to specific agents, agent clusters, or cloud agents. Wizards are available to assist you in calculating recommended capacity for specific agents.

The **Assign Agents** workflow bar button helps you get started with the following tasks:

- Configuring individual agents and adding them to the workbench agent pool
- Assigning individual agents to your project
- Assigning clusters of agents with pre-defined capabilities to your project
- Configuring your project to use agents that run as virtual machines in the cloud.

Assigning Agents to Workload

This task can only be performed after you have configured workload for your project.

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Click the **Agent Assignment** tab.
3. Select the **Assignment type**:

    - **Static assignment to project agents** : Use this method to statically assign specific agent computers (rather than clusters of agents) to your project. No agent-availability check is performed with this method and agent locking is disabled. Select this method if you want to use Agents deployed in the cloud.

- **Dynamic assignment to project agents** : With this method, workload is delivered using dynamic agent-assignment at execution time against the project's agents. Workload delivery is enhanced with agent-capability specifications to create optimized workload-to-agent assignments based on the capabilities of each agent. Agent locking at execution time is enabled with this method. Only responding agents that are not currently used by another controller are used with this method.
- **Dynamic assignment to Silk Central agent cluster** : Silk Performer workload delivered by way of Silk Central can also use dynamic workload-to-agent assignment. Within Silk Performer you choose the name of the agent cluster (from the drop list) that should deliver your test's workload. Silk Central then provides the list of agent computers that are assigned to the cluster. Workload is then assigned to specific agents at the moment of execution based on the capabilities of the individual agents. After you connect to Silk Central, you are presented with the list of available agent clusters. In the right-most window, you can view the agents that are currently associated with the selected agent cluster.

4. Define the agents that are to deliver the workload for your test.

   - If you selected **Static assignment to project agents** , you can check the **Even user distribution** check box to distribute all existing user types evenly across all agents, depending on each agent's general replay capabilities. To use agents that run as virtual machines in the cloud, check the **Use cloud agents** check box. Click **Cloud Agent Manager** to manage your agents in the cloud.
   - If you selected **Dynamic assignment to project agents** , workload is delivered automatically using dynamic agent-assignment at execution time against the project's agents.
   - If you selected **Dynamic assignment to Silk Central agent cluster** , you are asked to log in to Silk Central. When you are logged in, you can select the available agent cluster.

   The **Agents** list box displays the available agents.
5. Check the **Agent resource utilization** check box to assign a maximum percentage of total virtual users that each agent can run based on the agent's replay capabilities.
6. Check the **Balance load across agents** check box to apportion workload across agents.
7. If you selected **Static assignment to project agents** , use the lower window of the **Agent Assignment** page to define workload assignments for user groups.

   Note: Available options vary depending on the selected workload model.
8. Click **User Distribution Overview** to view the assignment of virtual users to the agent computers that are currently available and then click **Close**.
9. Click **OK** to save your settings.

Workload will be assigned to agents based on the agent-assignment settings you have configured. If there are not enough agents with the required capabilities to deliver the required workload, you will be presented with an error message and details regarding the user types that did not receive an agent assignment.

Agent Assignment Tab

The **Workload Configuration** > **Agent Assignment** tab enables you to configure how agents are assigned to a workload profile to deliver required load, including static or dynamic assignment to specific project agents and dynamic assignment to Silk Central agent clusters.

Item	Description
Agents	Lists the agents that are currently available for workload assignment along with the maximum virtual-user capacity that each agent can drive for each capability type (Java, ODBC, SAPGUI, etc.) Maximum virtual-user capacity depends on the **Agent Resource Utilization** setting. This list can assist you with error handling in cases of failed dynamic workload-assignment by displaying agents

Item	Description
	that can not be connected via the Silk Performer launcher service or are currently locked by other load tests. When logged in to the cloud, this list displays the available cloud agents side-by-side to other available agents.
Assignment type: Static assignment to project agents	Use this method to statically assign specific agent computers (rather than clusters of agents) to your project. No agent-availability check is performed with this method and agent locking is disabled. Select this method if you want to use Agents deployed in the cloud.
Assignment type: Dynamic assignment to project agents	With this method, workload is delivered using dynamic agent-assignment at execution time against the project's agents. Workload delivery is enhanced with agent-capability specifications to create optimized workload-to-agent assignments based on the capabilities of each agent. Agent locking at execution time is enabled with this method. Only responding agents that are not currently used by another controller are used with this method.
Assignment type: Dynamic assignment to Silk Central agent cluster	Silk Performer workload delivered by way of Silk Central can also use dynamic workload-to-agent assignment. Within Silk Performer you choose the name of the agent cluster (from the drop list) that should deliver your test's workload. Silk Central then provides the list of agent computers that are assigned to the cluster. Workload is then assigned to specific agents at the moment of execution based on the capabilities of the individual agents. After you connect to Silk Central, you are presented with the list of available agent clusters. In the right-most window, you can view the agents that are currently associated with the selected agent cluster. With this approach, the agents of a cluster can be changed without requiring modification of your Silk Performer project. Download of the agent clusters list from Silk Central requires Silk Central logon credentials. If your Silk Performer project is already associated with a Silk Central test definition, login is not required. Agent locking at execution time is enabled with this method. Only responding agents that are not currently in use by another controller are used with this method. If your project is not associated with a test definition in Silk Central, the Silk Central connection parameters from Silk Performer's system settings are used. When neither approach is successful, the **Download agent clusters from** Silk Central dialog appears. This dialog features the same values as those found at **Settings** > **System Settings** , Silk Central. Specify appropriate connection parameters and click **Finish**.
Even user distribution	This option is enabled when **Static assignment to project agents** is selected. Use this option to distribute all existing user types evenly across all agents, depending on each agent's general replay capabilities. For example, if you have 50 VUsers and 5 agents, each agent runs ten VUsers. If this option is not enabled, the first agent would run all 50 VUsers if replay capabilities permit. The other agents would rest idle, taking over the workload only if one agent reaches its maximum replay capability.

Item	Description
Use cloud agents	This option is enabled when **Static assignment to project agents** is selected. Use this option if you want to deploy agents that run as virtual machines in the cloud. When checking this checkbox, the cloud **Login** dialog box opens, asking for your user credentials (check **Save Credentials** to not be asked for your credentials again). Contact your sales representative to retrieve valid credentials.
Cloud Agent Manager	Click here to launch the Cloud Agent Manager, which allows you to manage your agents in the cloud.
Agent resource utilization	Check the **Agent resource utilization** check box to assign a maximum percentage of total virtual users that each agent can run based on the agent's replay capabilities. For example, if an agent can run 50 users of SAPGUI capability, then an **Agent resource utilization** value of `50%` results in 25 available SAPGUI virtual users for the test. This value is reflected in the **Agents** table.
Balance load across agents	This option is especially desirable for increasing or dynamic workloads. Note that this setting does not come into effect until the moment of execution. • Without this setting enabled, your user load may be distributed to agents sequentially, as follows: • Agent1: VU1, VU2, …, VU17 • Agent2: VU18, VU19, …, VU34 • Agent3: VU35, VU36, …, VU50 • With this setting enabled, virtual user distribution is more uniformly distributed across agents, as follows: • Agent1: VU1, VU4, VU7, … • Agent2: VU2, VU5, VU8, … • Agent3: VU3, VU6, VU9, …
UserType/Agents	This portion of the tab enables you to edit the users-to-agents assignments that are associated with your project. A roll-over tooltip on each usertype shows the capabilities that are required for the usertype. This is useful for error handling in cases when workload assignment fails.
User Distribution Overview	Click the **User Distribution Overview** button to view a summary of the users-to-agent assignments in your project.

Manually Assigning Agent Computers to User Types

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Click the **Agent Assignment** tab.
3. Define the agents that are to deliver the workload for your test.

   Click the **Static assignment to project agents** option button to statically assign specific agent computers (rather than clusters of agents) to your project. Agent locking is disabled with this method.
4. In the **Auto-assign** column, uncheck the **Auto-assign** check box.
5. Click **[...]** to the left of the **Agents** column. The **Manual Agent Assignment for Script/Usertype** dialog box opens.
6. Click the agent that you want to assign and click **OK**.

   All of the users from the user type must be assigned to agents.

The selected agent appears in the **Agent** column on the **Workflow Configuration** dialog box.

Dynamic Workload-Assignment

Silk Performer's dynamic workload-assignment functionality matches your specific load-test requirements to the replay capabilities of available agent computers at execution time. For example, if your load test requires a workload that can be delivered only by an agent computer with an installed SAPGUI client, dynamic workload-assignment functionality can ensure that your test's workload is assigned only to available agents with installed SAPGUI clients. Further, you can configure the percentage of required workload, in the number of virtual users, that must be allocated to each agent, thereby ensuring that agents are not pushed beyond their load capacities.

Dynamic-workload assignment allows you to account for variations in resource requirements. For example, approximately the same amount of resources is required to drive 2,000 virtual Web users over a Web replay interface as is required to drive 20 virtual SAP users over a SAPGUI replay interface. Based on maximum virtual-user-per-replay-capability settings that you configure, the dynamic-workload assignment algorithm allocates appropriate workload levels to agents. This approach does not necessarily result in each agent continuously driving its maximum supported load. You can define the percentage of maximum-potential workload that must be allocated to each individual agent within the cluster. For example, if an agent can drive 2,000 Web users and you set the agent-resource-utilization setting to `50%`, only 1,000 Web users are allocated to that agent. The remaining workload is allocated to other agents that possess the same Web-user capability.

An advantage of dynamic assignment of workload to agent clusters is the manner in which the successful execution of tests is not contingent on your maintaining a static test-execution environment. For example, if an agent's name changes or becomes unavailable, Silk Performer can dynamically assign the unavailable agent's workload to an available agent in the same cluster with the required capabilities. This approach eliminates many of the challenges involved in maintaining a static test environment and is of particular value when Silk Performer load tests are managed and executed based on predefined schedules in Silk Central. Silk Central can run tests against agent clusters rather than individual agents. Issues that do not require consideration from the Silk Central perspective include the health of individual agents and the manner in which workload is balanced across agents.

Agent computers remain available to all workloads until the moment of test execution because they are not reserved or assigned to any one specific workload. When a Silk Performer controller identifies an available agent on which it intends to run a test execution, the controller momentarily locks the agent to prevent conflict conditions with other controllers. After the test completes, the agents are unlocked and made available to other tests.

Defining the Capabilities of Agents

1. Choose **Settings** > **System** . The **System Settings** dialog box displays.
2. Click **Agents**. The **Agent Pool** tab displays.
3. Click the agent for which you want to define capabilities.
4. Click **Properties**.
5. Click the **Capabilities** tab.

   The **Capabilities** page is pre-populated with suggested replay-capability values that are based on the specifications of installed hardware.
6. Adjust the **Max. Vuser** values per capability type as required by the agent.

   You are responsible for setting appropriate maximum virtual-user values for your agents. Values from `0` to `9999` are valid.
7. Click **OK** to save your settings.

   Alternatively, you can click **Set Default** to restore default capability values.

Properties for Agent - Capabilities Tab

The **Properties for Agent** > **Capabilities** dialog enables you to configure the maximum number of virtual users per replay interface that the agent is capable of driving to optimize dynamic-workload assignment.

The **Capabilities** page is pre-populated with suggested replay-capability values that are based on the specifications of installed hardware. You are responsible for setting appropriate maximum virtual-user values for your agents. Values from `0` to `9999` are valid.

Capability	Max. Vusers
General	Maximum number of virtual users that the agent is capable of driving against replay interfaces that do not require any specific client installations on the agent. The `general` capability includes replay interfaces such as Web protocols and IIOP.
Browser-driven	Maximum number of virtual users that the agent is capable of driving against the browser-driven testing replay interface.
Java	Maximum number of virtual users that the agent is capable of driving against replay interfaces that require a Java Runtime installation. The `Java` capability includes replay interfaces such as Java Framework, OracleForms, Oracle Applications, Jolt, and Jacada.
ODBC	Maximum number of virtual users that the agent is capable of driving against the `ODBC` replay interface.
Oracle OCI	Maximum number of virtual users that the agent is capable of driving against the `Oracle OCI7` or `Oracle OCI8` replay interface.
SAPGUI	Maximum number of virtual users that the agent is capable of driving against the `SAPGUI` replay interface.
Citrix	Maximum number of virtual users that the agent is capable of driving against the `Citrix` replay interface.
Tuxedo	Maximum number of virtual users that the agent is capable of driving against the `Tuxedo` replay interface.
.Net	Maximum number of virtual users that the agent is capable of driving against the `.Net Framework` replay interface.
GuiL Test	Maximum number of virtual users that the agent is capable of driving against the `Gui-level testing` replay interface.
Default button	Click to reset all values to the suggested replay-capability values.

Replay Capabilities

You can tag each agent with the following replay capabilities:

- `General` – Any replay feature that does not require an installation (except browser-driven), including Web and IIOP.
- `Browser-driven` - Does not require an installation, but has a higher resource consumption capability than `General.`
- `Java` – A Java run-time environment is available for Java Framework, Oracle Forms, Oracle Applications, Jacada, and Jolt.
- `ODBC` – An ODBC client is installed.

- `Oracle OCI` – An Oracle client is installed.
- `SAPGUI` – A SAPGUI client is installed.
- `Citrix` – A Citrix client is installed.
- `Tuxedo` – A Tuxedo client is installed.
- `.Net` – The .NET Framework is installed.
- `GuiL Test` – Terminal Services is installed and running on this agent. Silk Test and the system under test must be installed.

For each capability, you can specify the maximum number of virtual users that the agent is capable of driving. Click the **Capabilities** tab (**Settings** > **System** > **Agents** > **Agent Pool** > **Properties** > **Capabilities** to complete this task.

Note: Depending on the hardware resources of the hosting machine, a load-test agent can replay a varying number of virtual users based on the resource demands of each particular replay interface. Therefore, agents are characterized by their hardware specifications, such as their CPU and memory, plus the number of virtual users that can be executed based on a particular replay interface.

Capability Complexity Configuration

Capability Complexities

Silk Performer uses abstract values, or *complexities,* to relate sets of capability resource requirements to one another. A capability is a technology that an agent is able to replay.

With dynamic workload-assignment, this is used for the following operations:

1. To sort agents by the sum of their capability complexities. This is used to assign user types to agents that have the lowest capability complexity first.
2. To sort user types by the sum of their associated scripts' capability complexity. This is used to assign user types that have the highest complexity first. Combined with operation 1 (above), this results in user types with the highest complexity being assigned to the agents that best fit the required capabilities.
3. To compute the default number of virtual users of an agent for each of its capabilities. For an agent, the number of virtual users for the `General` capability is computed based on the CPU speed, the number of CPUs, and available RAM. Using this value and the list of capability complexities, the number of available virtual users for all other capabilities is computed.

Example XML File

To override capability complexity values, you must create an XML file named `CapabilityComplexities.xml` and save it to `<public user documents>\Silk Performer 20.0`.

Such an XML file might look like this:

```
<Complexities>
  <Capability name="SAPGUI" complexity="150"/>
  <Capability name="ODBC" complexity="65"/>
</Complexities>
```

This example XML file overrides the complexity values for the SAPGUI and ODBC capabilities. All other values remain unchanged. The file can contain '0-n' Capability tags. The list of capability names is included in the table above.

Please note that specific complexity values are not that important. The important thing is what the values are *in relation* to each other. This has the consequence that if you were to multiply all complexities by the same factor, the relation of any two capabilities to each other would not change.

Setting a Default Number of Virtual Users

In some situations it is helpful to define a default number of virtual users for a capability (for example, to accommodate a technical constraint of the technology under test). This can be achieved by extending the capabilities defined in `CapabilityComplexities.xml` with `maxVU` attributes. For example, `<Complexities> <Capability name="GuiLTest" complexity="200" maxVU="30" /> </Complexities>`. The value specified in the `maxVU` attribute overrides the value that would otherwise be computed through the capability complexities. A `maxVU` value of `0` (which is the default when no `maxVU` attribute is specified) indicates that the default number of virtual users should be computed through the capability complexities.

Currently, the following fixed maximum VUser values are in effect by default:

- SAPGUI - 75 VUsers
- Citrix - 30 VUsers
- GuiLTest - 30 VUsers
- Browser-driven - 100 VUsers

Note: The maximum number of browser-driven virtual users is calculated as follows: 5% of capability, up to a maximum of 100.

Centralized Management of Load-Test Agent Clusters in Silk Central

Although you have the option of assigning workload to individual agents, assigning workload to clusters of agents is often preferable. *Clusters* are groups of agents that can share the same capabilities, such as a Java Development Kit (JDK) installation, an ODBC client, or a Citrix client, or can consist of agents that possess entirely heterogeneous capabilities, such as the capability to divide a lab's machines into separate agent groups that can support different teams. With this approach, you select a cluster of agents capable of executing the required workload when you configure your test's workload. Silk Performer then handles the dynamic assignment of workload to specific agents. Only those agents that are required to deliver the workload are actually used.

Agent clusters are managed centrally in Silk Central and are retrieved by Silk Performer when you start a load test—you do not need to configure the availability of agents yourself and conflicts with concurrently scheduled load tests are resolved automatically.

Creating a Silk Central Agent-Clusters File

Before you can complete this task you must export the Silk Performer agent pool as an XML file (which includes the connection properties, capabilities, and system information of an agent) for each agent in your agent pool.

You only need to create one Silk Central agent-clusters file. The file may contain one or more agent clusters, each of which specifies its associated agents including their capabilities, connection properties and system information. The contents of the file you create will be available to all Silk Performer users on the **Workload Configuration** dialog when they select the **Dynamic assignment to Silk Central agent cluster** option.

You must create a Silk Central agent-clusters file if you intend to run your test against a Silk Central agent cluster (this is configured by clicking the `Dynamic assignment to Silk Central agent cluster` button on the **Workload Configuration** > **Agent Assignment** tab). Workloads that use a Silk Central agent cluster can be executed from within the Silk Performer Workbench and they can be scheduled as Silk Central tests.

1. Create an empty XML file on your local system.

   This file must be accessible by Silk Central. It can be placed under source control within your Silk Central directory structure.
2. Use the contents of an exported agent pool file to build the agent-clusters file as structured in the example below.

   The contents of an exported agent-pool file and the agent-clusters file are nearly identical, so this typically only involves enclosing the `<AgentPool/>` elements of the exported agent-pool file within a `<SctmAgentClusters/>` element.

Example of a manually created Silk Central agent-clusters file

This manually created agent-clusters file includes a `SctmAgentClusters` root element. Within the root element are `AgentPool` elements, one for each agent pool in the cluster. Within each `AgentPool` element are `Agent` elements that convey the connection properties, capabilities, and system information of the individual agents.

```
<?xml version='1.0' encoding='UTF-8'?>

<SctmAgentClusters>
 <AgentPool name="cluster_1">
  <Agent id="LAB100">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.100</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="390" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="39" name="Citrix"></Capability>
    <Capability maxVU="3900" name="General"></Capability>
    <Capability maxVU="39" name="GuiLTest"></Capability>
    <Capability maxVU="390" name="Java"></Capability>
    <Capability maxVU="390" name="ODBC"></Capability>
    <Capability maxVU="390" name="Oracle OCI"></Capability>
    <Capability maxVU="39" name="SAPGUI"></Capability>
    <Capability maxVU="390" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3332</SysInfo>
    <SysInfo name="Memory">2039 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">1</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.0</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab101">
    <ConnProperty name="AuthPassword"></ConnProperty>
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="EncryptionSSL">false</ConnProperty>
    <ConnProperty name="HTTPTunnel">:8080</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.101</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="SOCKSTunnel">:1080</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <ConnProperty name="UseHttpTunnel">false</ConnProperty>
    <ConnProperty name="UseSocksTunnel">false</ConnProperty>
    <Capability maxVU="380" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="38" name="Citrix"></Capability>
    <Capability maxVU="3800" name="General"></Capability>
    <Capability maxVU="38" name="GuiLTest"></Capability>
    <Capability maxVU="380" name="Java"></Capability>
    <Capability maxVU="380" name="ODBC"></Capability>
    <Capability maxVU="380" name="Oracle OCI"></Capability>
    <Capability maxVU="38" name="SAPGUI"></Capability>
    <Capability maxVU="380" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3343</SysInfo>
    <SysInfo name="Memory">1983 MB</SysInfo>
    <SysInfo name="ProcType">Intel Pentium IV</SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack">Service Pack 2</SysInfo>
    <SysInfo name="SysVersion">5.2</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
 </AgentPool>
 <AgentPool name="cluster_2">
  <Agent id="LAB200">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.2.200</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="390" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="39" name="Citrix"></Capability>
    <Capability maxVU="3900" name="General"></Capability>
    <Capability maxVU="39" name="GuiLTest"></Capability>
    <Capability maxVU="390" name="Java"></Capability>
    <Capability maxVU="390" name="ODBC"></Capability>
    <Capability maxVU="390" name="Oracle OCI"></Capability>
    <Capability maxVU="39" name="SAPGUI"></Capability>
    <Capability maxVU="390" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3332</SysInfo>
    <SysInfo name="Memory">2039 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">1</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.0</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab201">
    <ConnProperty name="AuthPassword"></ConnProperty>
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="EncryptionSSL">false</ConnProperty>
    <ConnProperty name="HTTPTunnel">:8080</ConnProperty>
    <ConnProperty name="IpAddress">192.168.2.201</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="SOCKSTunnel">:1080</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <ConnProperty name="UseHttpTunnel">false</ConnProperty>
    <ConnProperty name="UseSocksTunnel">false</ConnProperty>
    <Capability maxVU="380" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="38" name="Citrix"></Capability>
    <Capability maxVU="3800" name="General"></Capability>
    <Capability maxVU="38" name="GuiLTest"></Capability>
    <Capability maxVU="380" name="Java"></Capability>
    <Capability maxVU="380" name="ODBC"></Capability>
    <Capability maxVU="380" name="Oracle OCI"></Capability>
    <Capability maxVU="38" name="SAPGUI"></Capability>
    <Capability maxVU="380" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3343</SysInfo>
    <SysInfo name="Memory">1983 MB</SysInfo>
    <SysInfo name="ProcType">Intel Pentium IV</SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack">Service Pack 2</SysInfo>
    <SysInfo name="SysVersion">5.2</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab203">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.2.203</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="650" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="65" name="Citrix"></Capability>
    <Capability maxVU="6500" name="General"></Capability>
    <Capability maxVU="65" name="GuiLTest"></Capability>
    <Capability maxVU="650" name="Java"></Capability>
    <Capability maxVU="650" name="ODBC"></Capability>
    <Capability maxVU="650" name="Oracle OCI"></Capability>
    <Capability maxVU="65" name="SAPGUI"></Capability>
    <Capability maxVU="650" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3371</SysInfo>
    <SysInfo name="Memory">3318 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">3000 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.2</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
 </AgentPool>
</SctmAgentClusters>
```

Once you have created an agent clusters file, you must configure Silk Central to reference the file. Silk Central will copy the file to the execution servers so that whenever a Silk Performer project with dynamic workload-assignment is executed, the project will read the file to determine how workload should be allocated to the agents within the cluster.

Exporting Agent Pool

Before you can create a Silk Central test-agent cluster file, you need to output the connection properties, capabilities, and system information of the available agents in your agent pool.

1. Navigate to **System** > **Settings** > **Agents** > **Agent Pool** .
2. Click the **Export Agent Pool** button.
3. Specify an appropriate path and filename to save the contents of your agent pool as an XML file.

Example agent pool file

This XML file includes an `AgentPool` root element. Within the root element are `Agent` elements, one for each agent in the agent pool. Within each `Agent` element are elements that convey the connection properties, capabilities, and system information of that agent.

```
<?xml version='1.0' encoding='UTF-8'?>
<AgentPool name="LocalPool">
  <Agent id="LAB108">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.108</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="390" name=".Net"></Capability>
    <Capability maxVU="100" name="Browser-driven"></Capability>
    <Capability maxVU="39" name="Citrix"></Capability>
    <Capability maxVU="3900" name="General"></Capability>
    <Capability maxVU="39" name="GuiLTest"></Capability>
    <Capability maxVU="390" name="Java"></Capability>
    <Capability maxVU="390" name="ODBC"></Capability>
    <Capability maxVU="390" name="Oracle OCI"></Capability>
    <Capability maxVU="39" name="SAPGUI"></Capability>
    <Capability maxVU="390" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3332</SysInfo>
    <SysInfo name="Memory">2039 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">1</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.0</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab116">
    <ConnProperty name="AuthPassword"></ConnProperty>
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="EncryptionSSL">false</ConnProperty>
    <ConnProperty name="HTTPTunnel">:8080</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.116</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="SOCKSTunnel">:1080</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <ConnProperty name="UseHttpTunnel">false</ConnProperty>
    <ConnProperty name="UseSocksTunnel">false</ConnProperty>
    <Capability maxVU="380" name=".Net"></Capability>
    <Capability maxVU="100" name="Browser-driven"></Capability>
    <Capability maxVU="38" name="Citrix"></Capability>
    <Capability maxVU="3800" name="General"></Capability>
    <Capability maxVU="38" name="GuiLTest"></Capability>
    <Capability maxVU="380" name="Java"></Capability>
    <Capability maxVU="380" name="ODBC"></Capability>
    <Capability maxVU="380" name="Oracle OCI"></Capability>
    <Capability maxVU="38" name="SAPGUI"></Capability>
    <Capability maxVU="380" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3343</SysInfo>
    <SysInfo name="Memory">1983 MB</SysInfo>
    <SysInfo name="ProcType">Intel Pentium IV</SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">3200 MHz</SysInfo>
    <SysInfo name="ServicePack">Service Pack 2</SysInfo>
    <SysInfo name="SysVersion">5.2</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab125">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.125</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="650" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
```

```
    <Capability maxVU="65" name="Citrix"></Capability>
    <Capability maxVU="6500" name="General"></Capability>
    <Capability maxVU="65" name="GuiLTest"></Capability>
    <Capability maxVU="650" name="Java"></Capability>
    <Capability maxVU="650" name="ODBC"></Capability>
    <Capability maxVU="650" name="Oracle OCI"></Capability>
    <Capability maxVU="65" name="SAPGUI"></Capability>
    <Capability maxVU="650" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3371</SysInfo>
    <SysInfo name="Memory">3318 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">3000 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.2</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
  <Agent id="lab47">
    <ConnProperty name="ConnectPort">19200</ConnProperty>
    <ConnProperty name="ConnectSecurePort">19201</ConnProperty>
    <ConnProperty name="IpAddress">192.168.1.47</ConnProperty>
    <ConnProperty name="LastConnectStatus">5</ConnProperty>
    <ConnProperty name="UseAuthentication">false</ConnProperty>
    <Capability maxVU="180" name=".Net"></Capability>
                <Capability maxVU="100" name="Browser-driven"></
Capability>
    <Capability maxVU="18" name="Citrix"></Capability>
    <Capability maxVU="1800" name="General"></Capability>
    <Capability maxVU="18" name="GuiLTest"></Capability>
    <Capability maxVU="180" name="Java"></Capability>
    <Capability maxVU="180" name="ODBC"></Capability>
    <Capability maxVU="180" name="Oracle OCI"></Capability>
    <Capability maxVU="18" name="SAPGUI"></Capability>
    <Capability maxVU="180" name="Tuxedo"></Capability>
    <SysInfo name="AgentRAC">7803300</SysInfo>
    <SysInfo name="AgentVersion">7.8.0.3414</SysInfo>
    <SysInfo name="Memory">1007 MB</SysInfo>
    <SysInfo name="ProcType"></SysInfo>
    <SysInfo name="ProcessorCount">2</SysInfo>
    <SysInfo name="ProcessorSpeed">2593 MHz</SysInfo>
    <SysInfo name="ServicePack"></SysInfo>
    <SysInfo name="SysVersion">5.1</SysInfo>
    <SysInfo name="System">WinNT</SysInfo>
  </Agent>
</AgentPool>
```

You can now use the XML data in the exported agent-pool file to create a Silk Central test-agent clusters file.

Assigning Workload to Cloud Agents

This task can only be performed after you have configured workload for your project.

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Click the **Agent Assignment** tab.
3. Select the **Assignment type**:
    - **Static assignment to project agents** : Use this method to statically assign specific agent computers (rather than clusters of agents) to your project. No agent-availability check is performed with this method and agent locking is disabled. Select this method if you want to use Agents deployed in the cloud.

4. Define the agents that are to deliver the workload for your test.
   - If you selected **Static assignment to project agents** , you can check the **Even user distribution** check box to distribute all existing user types evenly across all agents, depending on each agent's general replay capabilities. To use agents that run as virtual machines in the cloud, check the **Use cloud agents** check box. Click **Cloud Agent Manager** to manage your agents in the cloud.

   Note: To access Micro Focus CloudBurst services, register on *http://cloud.borland.com* or contact your sales representative.
5. Check the **Agent resource utilization** check box to assign a maximum percentage of total virtual users that each agent can run based on the agent's replay capabilities.
6. Check the **Balance load across agents** check box to apportion workload across agents.
7. If you selected **Static assignment to project agents** , use the lower window of the **Agent Assignment** page to define workload assignments for user groups.

   Note: Available options vary depending on the selected workload model.
8. Click **User Distribution Overview** to view the assignment of virtual users to the agent computers that are currently available and then click **Close**.
9. Click **OK** to save your settings.

Note: Cloud agents are considered 'unreachable' when the Cloud Agent Manager cannot connect to them. There are a few possible reasons why this may occur:

- Your network contains a firewall that blocks certain hosts or ports. Agent communication occurs over a secure HTTP tunnel using port 443.
- Your network uses a proxy server that blocks certain Web sites or ports.
- The machine that the Cloud Agent Manager runs on has a firewall that blocks certain hosts or ports.

Defining Number of VUsers per Cloud Agent

1. Click **Assign Agents** on the workflow bar. The **Workflow - Assign Agents** dialog box appears.
2. Click **Use Cloud Agents**. The **Workflow - Prepare Cloud Agents** dialog box appears.
3. Slide the slider to the right to select the number of VUsers that you want to have run on each cloud agent.

   Markers along the slider indicate the number of VUsers that can typically be supported by cloud-based agents when testing various application types. However, the maximum number of VUsers per cloud agent, regardless of application type, is 1,000. The number of VUsers that can actually be supported for your testing scenario may vary of course, depending on the complexity and resource consumption of your specific test.

   The **Workload** text field shows the maximum number of VUsers that have been defined for each user type (as defined on the **Workflow - Workload Configuration** dialog box). The **Assignment per region** table shows the resulting allocation of agents across all geographic regions based on your selection. For example, with a **Workload** setting of 100 VUsers and a setting of 20 max VUsers per agent across five geographic regions, each region will run 20 VUsers. The higher the maximum VUsers per agent setting, the fewer agent instances required within each geographic region to support the prescribed workload.
4. Click **Next**. The **Start Agent(s)** dialog box appears showing you the number of agent instances within each geographic region that will be started to deliver the prescribed workload.
5. Within the **Up Time** area of the dialog box, specify how long you want the agent instances to remain active. Select **Forever** or a specific number of **days** and/or **hours**.

   The **Company Limit** shows the remaining number of agent instances that you can start based on your license type.
6. Click **Start** to open Cloud Agent Manager and start the specified number of cloud agent instances.

Evaluating Agent VUser Capacity

If you are not sure how many virtual users your agent can handle, you can perform an evaluation run. Once the evaluation run is finished, Silk Performer gives you an estimation of how many virtual users you can run on the specified agent.

On the **Evaluate Agent VUser Capacity** dialog, click **Start**. Silk Performer will gradually increase the number of virtual users. The run is automatically stopped if at least one of the following is true:

- The responsiveness falls below 95%.
- The CPU usage exceeds 95%.
- The maximum number of virtual users is reached.

Note: If you want to evaluate the capacity of a cloud agent, you need to have Micro Focus Credits on your cloud account.

Trying Out Agents

Before you start your load test, it can be useful to verify that the test script you created works on all agents you are planning to use for the load test.

Try Agents Settings

For Try Agents runs, the following options are automatically set to these specified values (see also "Replay Options"):

- The **Stress test** option is on, when think times are disabled.
- The **Stop virtual users after simulation time (Queuing Workload)** option is off.
- The **Virtual user log files (.log)** option is on.
- The **Virtual user output files (.wrt)** option is on.
- The **Virtual user report files (.rpt)** option is on.
- The **Virtual user report on error files (.rpt)** option is on.
- The **TrueLog files (.xlg)** option is off.
- The **TrueLog On Error files (.xlg)** option is on.
- The **Compute time series data (.tsd)** option is off.
- All logging detail options (**Results** > **Logging** and **Results** > **Internet Logging** page) are on.
- The **Enable all measure groups (TSD measure groups)** option is off.
- The **Bandwidth** option is set to `High Speed (unlimited)`.
- The **Downstream** option is set to `unlimited`.
- The **Upstream** option is set to `unlimited`
- The **Duplex** option is off.

Trying Out a Test Script On Agents

You must record or manually create a test script before you can try out your script on various agents.

1. Click the **Try Agents** button on the Silk Performer Workflow bar. The **Workflow - Try Agents** dialog box appears.
2. Select one or more user types from the list **User Types to execute**.

   Each user type will be executed on each selected agent. For example: If you select two user types and three agents, Silk Performer will start six test runs in total.
3. Select one or more agents from the list **Agents**.

   Note: You can select local agents and cloud agents, or more specifically: cloud regions. If you select a cloud region, all cloud agents from the respective region will be tested. It is not possible to test individual cloud agents. You can start cloud agents in the **Cloud Agent Manager**. Note that it can take some time until the cloud agents are ready to execute tests. A note beside the cloud regions tells you how many cloud agents are reachable and ready for a test execution. For example: **2/4 Agents ready** means that 4 agents were started in the **Cloud Agent Manager** and 2 of these are ready for execution.

4. *Optional:* Click **Enable think times** if you want to consider the think times in your script during the run. This option is disabled by default.

   Note: Usually, a Try Agent run is used to verify that a test script works correctly on various agents. For such a functional test, think times can be neglected, since they are a means to create a more realistic user behaviour and therefor a more realistic load. However, load issues can be neglected in a functional test.
5. Click **Run** to try out the script on the specified agents.
6. If you have selected cloud regions to be tested, the **Review Estimated Micro Focus Credits Consumption** dialog box displays. It gives you an estimation of how many Micro Focus Credits the runs will consume. Click **Accept and Run**.

The **Monitor** window opens, giving you detailed information about the progress of the Try Agents run. Once all runs are finished, the **Try Agents Summary** displays.

Try Agents Summary

When a Try Agents run is complete, the **Try Agents Summary** page appears. You can also open this page from the **Results** tree. You can perform the following actions in the **Next Steps** area on the right side:

- Click **Analyze Errors** to view the errors in TrueLog Explorer (if any occurred).
- Click **Configure Monitoring** to continue with the next step in the workflow bar.
- Click **View debit information** to show the amount of Micro Focus Credits that was debited from your account.
- Click **Local Agents** or one of the cloud regions in the **Analyze Result Files** area to get detailed result information about the runs that were executed on the various agents.

Note: If you want to prevent the summary page to appear each time a test is complete, disable the **Show Summary Page** button in the toolbar of the **Monitor** page.

Configuring Monitoring

Before running a test you need to define how Performance Explorer, the Silk Performer server monitoring tool, is to monitor local and remote servers involved in your test. Server monitoring reveals, locates, and assists in resolving server bottlenecks, allowing you to examine the performance of operating systems and application servers.

Three monitoring options are available:

- **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
- **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.
- **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

Server Monitoring

During a load test, Silk Performer provides for server monitoring by allowing you to view a live, graphical display of the server performance while a test runs.

The monitoring of servers enables the generation of server-side results that can be viewed and correlated with other test measurements at the results exploration stage. Monitoring also helps you learn whether bottlenecks exist on the server and, if so, their exact location. As a result, monitoring lets you examine the performance of both the OS and the server application.

You can set up a template for server monitoring to monitor performance data, or you can use a default template that is already generally configured for the type of application you are testing.

Defining Monitoring Options

1. Click **Configure Monitoring** on the workflow bar. The **Workflow - Configure Monitoring** dialog box appears.
2. Select one of the following options and click **Next**:

   - **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
   - **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.
   - **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

   (for Default Monitoring and Custom Monitoring only) A confirmation dialog box will notify you if you have logging enabled. Logging may skew your test results.
3. Click **OK** to accept your logging settings or click **Cancel** to adjust your logging options (**Settings** > **Active Profile** > **Results** > **Logging**).
4. *(for Custom Monitoring only)* Performance Explorer starts and the **Data Source Wizard** opens. Complete the steps outlined in the wizard.
5. The **Workflow - Workload Configuration** dialog box appears. Click **OK** to accept your monitoring settings.

Setting Up a Template for Server Monitoring

1. Go to **Settings** > **Active Profile** The **Profile Settings** dialog box opens.
2. Click the **Results** icon in the group box.
3. Click the **Monitoring** page.
4. In the **Monitoring options** area, check the **Automatically start monitoring** check box to automatically launch Silk Performer's monitoring facility when the test starts.
5. To automatically use the monitoring template that best suits the project, click the **Use default monitoring template** option button.
   For example, if you are creating a Web project, the template specifies the measurements that are useful for Web load tests.
6. To use a custom monitor template, click the **Use custom monitoring template** option button and perform one of the following steps:

    - Type the name of the custom template file (`.pew`) that you want to use to monitor your server. Silk Performer creates a copy of the standard monitor template.
    - Click the folder icon in the name field to select an existing monitoring template.
7. *Optional:* Click **Edit Custom Monitor Template** to add or remove any monitoring performance data.

   When you click this button, Silk Performance Explorer opens. Perform the following steps:

   a) Add or remove any monitoring performance data.
   b) Save the Silk Performance Explorer workspace to apply your changes to the template.
8. In the **Performance Monitor integration** area, check the **Compute online performance data** check box to compute data for additional performance measurements to be displayed in the Windows Performance Monitor.

   You can use this data to view concurrent users, transaction throughput, sent and received data, and executed SQL statements.
9. Click **OK**. After you start a load test, server monitoring for the load test starts and stops automatically.
10. To save monitoring results for future exploration, write the server-monitoring results to a monitoring writer file: In Performance Explorer, on the **Real-Time Monitoring** tab, in the **New** group, click **Monitor Writer**.

Running Load Tests

Part of the process of conducting a Silk Performer load test is to run a full load test.

Running Tests

In a load test, multiple virtual users are run by means of a test script against a target server to determine the load impact on server performance. A large load test requires an appropriate testing environment on your LAN, including a full complement of agent computers to host the virtual users.

It is essential that you complete the following tasks:

- Set options for the appropriate type of test that is to be run
- Accurately define the required workloads
- Enable the generation of test results to assess server performance

Do not enable complete result-logging during load testing because it might interfere with load-test results. However, the **TrueLog On Error** logging option writes necessary log files to a disk when errors occur, allowing you to inspect replay errors visually.

Real-time information regarding agent computers, virtual users, and transactions is displayed for you while load tests run. Real-time performance monitoring of the target server is presented in graphical format.

When a test is configured as a verification run, the following options are automatically set to the specified values:

- A **Baseline report** file is automatically created.
- The **Stop virtual users after simulation time (Queuing Workload)** option is disabled.
- The **Virtual user log files (.log) option** is disabled.
- The **Virtual user report files (.rpt)** option is enabled.
- The **Virtual user report on error files (.rpt)** option is enabled.
- The **Compute time series data (.tsd)** option is disabled.

Running a Load Test

Run a load test after you set up your testing environment and configure all test settings.

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Configure the workload that you plan to use in your load test.

3. *(Optional)* Click **Connect** on the **Workload Configuration** dialog box to initialize the agent connection without starting the test.
   a) Click **OK** on the **New Results Files Subdirectory** dialog box.
   b) Click **Start all** on the Silk Performer toolbar to manually start the test from monitor view.
4. Click **Run** to start the load test.
5. Click **OK** on the **New Results Files Subdirectory** dialog box.

   (Optional) To specify a name for the results subdirectory, uncheck the **Automatically generate unique subdirectory** check box and enter a name for the new subdirectory in the **Specify subdirectory for results files** text box.

Monitor test progress and server activity by viewing the Silk Performer tabular monitor view and the Performance Explorer graphical monitor view.

Calculating Virtual Users

On the **Workload Configuration** dialog, on the **Pacing** tab, you can perform the following actions:

- You can let Silk Performer calculate the **Max. VUsers**
- You can let Silk Performer calculate the **Sessions Per Peak Hour**
- You can let Silk Performer calculate the **Goal Trans Per Second**
- You can set your own specific **Goal Session Time** by enabling pacing

Tip: You can copy and paste the values in the **Workload Configuration** grids by using the keyboard shortcuts **Ctrl+C** and **Ctrl+V** or by right-clicking on a value and using the buttons in the context-menu.

Session Pacing

A *session* is the group of transactions defined for a user in the `dcluser` section. In the following sample, the session consists of 4 transactions. Note that begin and end transactions are not part of the session. The *session time* is the time it takes to execute the regular transactions including think times.

```
// Workload Section
dcluser
  user
    VirtUser
  transactions
    TInit      : begin;
    TLogin     : 1;
    TAddToCart : 2;
    TCheckout  : 1;
    TEnd       : end;
```

The *baseline session time* shows how long it took to execute the session in the baseline test.

Session pacing causes Silk Performer to add additional wait time, the so-called pacing wait time, at the end of a session before a new one is started. Session pacing allows you to reach a specified fixed session length, the so-called *Goal Session Time*.

Typically, you have already defined the number of virtual users in the **Adjust Workload** workflow step. In this table, you can now see how many sessions the virtual users will manage to execute in one hour. This is reflected by the **Sessions Per Peak Hour** value. If you adjust this value, Silk Performer will calculate how many virtual users you need to reach the number of sessions. On the other hand, if you adjust the number of virtual users, Silk Performer will recalculate the number of sessions the virtual users will manage to execute.

How the Sessions Per Peak Hour value is calculated

- Make sure you have set a baseline. See *Finding Baselines*.

- Select to **Calculate virtual users required to reach a certain number of** `sessions per hour`.
- The **Baseline Session Time [s]** column now contains the session time of each user type in the baseline.
- The **Sessions Per Peak Hour** is calculated for each user type.

Here is an example of how Silk Performer calculates the values: If the baseline test was executed in 100 seconds and you configured 10 virtual users, these 10 virtual users will be able to execute about 360 sessions within one hour (10 x 3600 seconds / 100 seconds).

```
Session Per Peak Hour = (Virtual Users x 3600 seconds) / Baseline Session Time
```

How to set your own specific Goal Session Time

As outlined above, Silk Performer uses the baseline session time to calculate the Sessions Per Peak Hour and the Max. VUsers. If you do not want to use the given baseline session time but want the sessions of a user type to take a different time, you can set your own *Goal Session Time* by configuring a pacing wait time.

How to configure Session pacing

- Select `sessions per hour` to calculate the virtual users required to reach a certain number of sessions per hour.
- If you have set a baseline previously, the **Baseline Session Time [s]** column now contains the session time value of each user type in the baseline. Your inputs in the subsequent steps will be validated against the baseline.
- In the **Pacing** column, click **...** to open the **Configure pacing** dialog.
- Click **Wait time insertion**.
- Select the relative **Pacing wait time** or the absolute **Goal Session Time** and specify the value in the fields beneath the chart.
- Click **OK**. Your **Goal Session Time** displays in the table and the **Max. VUsers** value is adjusted. The **Bandwidth** value updates if a baseline has been set.

How to configure Think Time adaption

- Select `sessions per hour` to calculate the virtual users required to reach a certain number of sessions per hour.
- If you have set a baseline previously, the **Baseline Session Time [s]** column now contains the session time value of each user type in the baseline. Your inputs in the subsequent steps will be validated against the baseline.
- In the **Pacing** column, click **...** to open the **Configure pacing** dialog.
- Click **Think time adaptation**.
- Select **Static** or **Dynamic**:
  - Static: To reach the goal session time, the think times in the script are multiplied with a constant, static factor.
  - Dynamic: To reach the goal session time, the think times in the script are multiplied with a dynamic factor that is recalculated after each executed session.
- Define your **Goal Session Time** and click **OK**. Your **Goal Session Time** displays in the table and the **Max. VUsers** value is adjusted. The **Bandwidth** updates if a baseline has been set.

Transaction Pacing

While *Session Pacing* is adding pacing wait time at the end of the session, *Transaction Pacing* is adding pacing wait time at the end of each individual transaction.

How to get the Goal Transactions Per Second calculation

- Select `transactions per second` to calculate the virtual users required to reach a certain number of transactions per second.
- If you have set a baseline previously, the **Avg Transaction Time [s]** column now contains the average transaction time value of each user type in the baseline.
- The **Goal Trans Per Second** column is calculated for each user type row.

How to set your own specific Goal Transactions Per Second

- Select `transactions per second` to calculate the virtual users required to reach a certain number of transactions per second.
- If you have set a baseline previously, the **Avg Transaction Time [s]** column now contains the transaction time value of each user type in the baseline.
- In the **Pacing** column, click **...** to open the **Configure pacing** dialog.
- Click **Wait time insertion**.
- Select the relative **Pacing wait time** or the absolute **Avg Goal Trans Time** and specify the value in the fields beneath the chart.
- Click **OK**. Your **Goal Session Time** displays in the table and the **Max. VUsers** value is adjusted. The **Bandwidth** updates if a baseline has been set.

Pacing

A commonly used load testing model is to keep transaction rates constant regardless of how loaded the server system is already. In Silk Performer you can define workloads based on transaction or session rates, which Silk Performer tries to meet even when server response times increase due to the increasing load. Consequently, when server response time slows, Silk Performer automatically reduces think times or pacing wait times to keep session times constant.

Silk Performer offers two methods to keep transaction or session rates at a defined level.

- **Think time adaptation** modifies the think times in your script so that a certain session time is reached.
- **Pacing wait time** is the period of time that a virtual user idles after a transaction or session has been completed, so that a certain transaction or session rate is reached.

To modify pacing options, click **Run Test** on the workflow bar and click the **Pacing** tab.

To open the **Configure Pacing** dialog, click **...** in the **Pacing** column.

Several options are available to configure pacing:

- **Think time adaptation**: Click this option to determine a **Goal Session Time**. The think times will be automatically adjusted to match the specified goal session time. You can select between **Static** and **Dynamic**.
- **Wait time insertion**: Specify either the **Pacing wait time** or the **Goal Session Time**. If you specify the pacing wait time, it will be added to the execution time of a transaction. If you add up these two times, you will get the goal session time. If you specify the goal session time, the pacing wait time will be automatically determined. You can also specify to randomize the entered values.

To configure pacing wait time directly in a BDL script, use the BDL functions `SetPacingTime()` and `GetPacingTime()`.

Silk Performer offers pacing for the following workload models:

- Increasing
- Steady State
- All Day
- Dynamic

Monitoring Load Tests

Detailed real-time information is available to testers while Silk Performer load tests run. Graphic displays and full textual reporting of activity on both the client side and the server side offer intuitive monitoring of the progress of tests as they occur.

Directly from the workbench on which the test is conducted, a tester can view comprehensive overview information about the agent computers and virtual users in the test. A tester can control the level of detail of the displayed information, from a global view of the progress of all the agent computers in the test to an exhaustive detail of the transactions conducted by each virtual user. Progress information for each agent and each user is available in multiple categories. Run-time details for each user include customizable, color-coded readouts on transactions, timers, functions, and errors as they occur.

In addition, real-time monitoring of the performance of the target server is available in graphical form. Charts display the most relevant performance-related information from a comprehensive collection of the Web servers, application servers, and database servers running on all of the OSes most widely used today. Multiple charts can be open at the same time, and these charts can be juxtaposed to provide the most relevant comparisons and contrasts for the tester. A menu tree editor allows for the combination of elements from any data source in the charts. Response times and other performance information from the client application can be placed in the same chart as performance data from the server. This feature enables a direct visual comparison so that one can deterimine the influence of server shortcomings on client behavior.

Automatic Monitoring

If **Automatically start monitoring** is enabled in the Silk Performer profile when a load test begins, Silk Performer launches Performance Explorer with the monitoring template you assigned to the profile.

All monitor writers begin writing and saving generated TSD files to the load-test directory. The writing of these monitor files is automatically stopped after load tests are complete.

The names of generated TSD files are formatted as `r@NAMEOFMONITOR@STARTTIME.tsd`, where `NAMEOFMONITOR` is the monitor writers caption. After running the load test, TSD files for each monitor writer are defined in your workspace.

Monitoring All Agent Computers During Load Testing

Use the **Monitor** window to view progress while a load test runs. The top part of the window displays information about the progress of agent computers and user groups.

Among the comprehensive number of information options, you can view the following information:

- Status of a particular agent
- Percentage of the test that is complete on an agent
- Number of executed transactions

Monitoring a Specific Agent Computer During Load Testing

In the top part of the **Monitor** window, select the specific agent to monitor.

The following information about the virtual users running on the selected agent appears in the bottom of the **Monitor** window:

- Status
- Name of the current transaction
- Percentage of work completed
- Number of executed transactions

Monitoring a Specific Virtual User During Load Testing

In the bottom part of the **Monitor** window, right-click the virtual user that you want to monitor and choose **Show Output of Vuser**.

In the **Virtual User** window, Silk Performer displays detailed run-time information about the selected user, such as the transactions and functions the user executes and the data the user sends to and receives from the server.

Tip: Right-click the virtual user area and choose **Select Columns** to select the columns you want to view.

Using a Graph to Monitor Server Performance

1. Click **Confirm Baseline** on the workflow bar. The **Workflow - Confirm Baseline** dialog box opens.
2. Click **Define monitoring options** to specify the settings for receiving online performance data. The **Profile Results** dialog box opens.
3. In the **Profile Results** dialog box, check the **Automatically start monitoring** check box to automatically start monitoring while running a load test and then choose one of the following options:
    - Click the **Use default monitoring template** option button.
    - Click the **Use custom monitoring template** option button to create a customized monitoring template.
4. Click **Create/Edit Custom Monitor Template**. Performance Explorer appears.
5. Close all monitor windows that you are not currently using.
6. Click **Monitor Server** on the Performance Explorer workflow bar.

   Alternatively, you can choose **Results** > **Monitor Server** from the menu bar.

   The **Data Source Wizard / Select Data Sources** dialog box opens.
7. Perform one of the following steps:
    - If you are certain of the data sources that the server provides, click the **Select from predefined Data Sources** option button to then select them from the list of predefined data sources.
    - If you are uncertain of the data sources that the server provides, click the **Have Data Sources detected** option button to let Performance Explorer scan the server for available data sources.
8. Click **Next**.

   In the menu tree, expand the folder that corresponds to the OS on which the server and application are running.
9. Select the server application you want to monitor from the list that appears.
   For example, to monitor the OS, select **System**.
10. Click **Next**. The **Connection Parameters** dialog box opens.
11. In the **Connection parameters** area, specify connection parameters such as the host name or IP address of the appropriate server system, the port number, and other data required to connect to the data source.

    The specified data depends on the OS running on the computer that you are monitoring.
12. Click **Next**. The **Select Displayed Measures** dialog box opens.
13. Expand the menu tree and select the factors you want to monitor.
14. Click **Finish**. A Monitor graph shows the specified elements in a real-time, color-coded display of the server performance. Beneath the graph is a list of included elements, a color-coding key, and performance information about each element.

Monitoring Performance for Multiple Servers

For each server that you want to monitor, follow the procedure for monitoring server performance by using a graph. Separate graphs appear for each server.

1. Reposition and resize the graphs to facilitate viewing and comparison.
2. To monitor factors from multiple servers in a single graph, use the menu tree to locate the appropriate factors and drag them onto the selected graph.

Using a Report to Monitor Server Performance

Complete the steps for using a graph to monitor server performance.

1. Choose **Monitor** > **Clone Monitor Report** from the Performance Explorer menu bar. A Monitor report displays the performance factors that were selected in the graph. Monitoring information now appears in tabular form in the report.
2. To save the monitoring report for results exploration along with the load test results, choose **Monitor** > **Write Monitor Data** from the Performance Explorer menu bar.

Storing Server-Monitoring Data

Create a Monitor writer and populate it with the relevant performance factors.

Note: You can store server-monitoring data from a Monitor writer but not from a Monitor chart.

1. In Performance Explorer, on the **Real-Time Monitoring** tab, in the **New** group, click **Monitor Writer**. The file name appears in the **File** section of the **Monitor information** area in the report dialog box. The monitor data is stored in a time series data (`.tsd`) file.
2. To view the data in a chart, click **Monitor Chart** in the **New** group on the **Real-Time Monitoring** tab. A new Monitor chart appears.
3. Drag the appropriate performance factors from the writer onto the chart. The data appears in graphical form.

Starting Silk Performer from the Command Line

Command Syntax

You can start Silk Performer load tests directly from a command-line interface. This is especially useful when you create a batch file, which executes a number of load tests consecutively.

To start Silk Performer from a command line, use the following syntax:

```
performer [project file [/Automation [refresh rate] [/WL:workloadname] [/
StartMonitor]] [/Resultsdir:directory]]
```

Here is a sample command with sample parameters:

```
performer "<my documents>\Silk Performer <version>\Projects\AllDayWorkload
\AllDayWorkload.ltp" /Automation 5 /WL:AllDayWorkload10
```

Note: When files or folders in the project path contain spaces, include quotation marks (") around the project path.

Note: Silk Performer will close immediately if an error (such as a misspelled project path, script error, workload not in the specified project file) occurs during the execution of the command line instruction. Errors can be found in the Event Viewer of the operating system.

Command Line Parameters

Parameter	Description
`project file`	Name of the project file that defines the load test to be executed.

Parameter	Description
`/Automation`	Automatically starts the load test that the specified project file defines. If you omit this parameter, Silk Performer opens the specified project.
`refresh rate`	Refresh rate, in seconds, at which the **Monitor** window refreshes. If you omit this parameter, then no refresh occurs.
`/WL:workloadname`	Name of the workload in the specified project file. If you specify a workload, it is set as the active workload in the project file.
`/StartMonitor`	Starts Performance Explorer’s monitoring facility when running the test.
`/SOA`	Launches the Silk Performer SOA Workbench.
`/Vision`	Specifies to start as Silk Performer Monitor Workbench.
`/eCatt`	Specifies to start as Silk Performer SAPGUI eCatt Workbench.
`/Abort`	Aborts running tests.
`/? or -?`	Displays help about the Silk Performer command-line interface.
`/Resultsdir:directory`	Specifies the directory where the load test results are stored. The directory path can be absolute or relative to the project file.
`/ImportStAsset:asset file?project name`	Imports Silk Test assets to Silk Performer. `Asset file` is the absolute path to the asset. `Project name` is the name for the new project. This value is optional. When omitted, the default project name is used.

Exploring Test Results

Usually the final step in conducting a Silk Performer load test is to display and analyze the results of the test to assess the performance of the application and the server under test.

Results Overview

The **Results** tab in the bottom left area of the GUI offers full control over your test results. Selecting the **Results** tab displays a menu tree containing all test results of the currently active project, including Try Script runs, load test results, and Silk Central results (if your Silk Performer project is linked to a Silk Central test definition).

Hovering your mouse over a test in the **Results** menu tree displays a summary of the test run, including date and time, the selected workload model, agent, VUser and error information.

Depending on the logging options that you have set for test runs, the following items are displayed in the **Results** menu tree:

- Baseline Report
- Time Series
- TrueLog Files (contains TrueLog files for each virtual user)
- User Results (contains user profiles, which list all result files per virtual user)
- Silk Central (displays results from Silk Central, if the project is linked to a Silk Central test definition)

Opening Results Files

Choose one of the following methods to open a results file:

- Double-click a file in the **Results** menu tree.
- Right-click a result file and select **Open** or **Explore**.

Opening a results file invokes the respective viewing facility.

Note: If the icon of a parent node in the **Results** menu tree is marked with a red X symbol, an error has occurred during the test execution. If an icon (parent node or result file) is marked with a blue exclamation mark (!), an error log file contains an informational message, a warning, or an error.

Deleting Locally Stored Results Files

Right-click a top-level results node and select **Delete Results**.

Note: This does not affect results that are stored in Silk Central.

Refreshing Displayed Results

Right-click a top-level results node and select **Update Results**.

The results that are displayed in the menu tree do not update if they are changed outside of Silk Performer.

Note: If the icon of a parent node in the **Results** menu tree is marked with a red X symbol, an error has occurred during the test execution. If an icon (parent node or result file) is marked with a blue exclamation mark (!), an error log file contains an informational message, a warning, or an error.

Silk Central Integration

Silk Performer is tightly integrated with Silk Central. If your Silk Performer project is linked to Silk Central, you can store your test results centrally in Silk Central where they can be accessed from both Silk Central and Silk Performer.

To view Silk Central results, your project must be linked to a Silk Central test definition . Once your project is linked to Silk Central, you can view the centrally stored results by clicking **Click here** to add Silk Central results node in the **Results** menu tree.

Additionally, you can upload results to Silk Central by right-clicking a top-level results node and selecting **Upload Results to Silk Central**.

Exploring Results

Silk Performer offers several approaches to displaying, reporting, and analyzing test results. Defined measurements take place during tests and can be displayed in a variety of graphical and tabular forms. Options include the following:

- **Performance Explorer**: This is the primary tool used for viewing test results. A fully comprehensive array of graphic features displays the results in user-defined graphs with as many elements as are required. The results of different tests can be compared. There are extensive features for server monitoring. A comprehensive HTML based overview report that combines user type statistics with time series test result information is also available.
- **TrueLog On Error:** Silk Performer provides full visual verification under load capabilities for various application types. It allows you to combine extensive content verification checks with full error drill-down analysis during load tests.
- **Virtual User Report files:** When enabled, these files contain the simulation results for each user. Details of the measurements for each individual user are presented in tabular form.

- **Virtual User Output files:** When enabled, these files contain the output of write statements used in test scripts.
- **Baseline Reports:** A detailed XML/XSL-based report that provides you with a summary table, transaction response-time details, timers for all accessed HTML pages, Web forms, and errors that occurred. This information is available for all user types involved in baseline tests.
- **Silk Central Reports:** Silk Performer projects can be integrated into Silk Central (Silk Central) test plans and directly executed from Silk Central. This allows for powerful test-result analysis and reporting. For detailed information on Silk Central reporting, refer to *Silk Central Help*.

Load Test Summary

When a load test run is complete, the **Load Test Summary** page appears. You can also open this page from the **Results** tree or by clicking **Explore Results** on the workflow bar. The **Load Test Summary** page contains:

- a **Quick Summary**, which gives you an overview about the test duration, users, agents, and errors.
- an **Available User Types** area, which you can use to drill down on the results for each user type. Select a user type from the list.

You can perform the following actions in the **Next Steps** and **Analyze Result Files** area on the right side:

- Click **Analyze load test** to view all detailed metrics in Performance Explorer.
- Click **Analyze errors** to view the errors in Performance Explorer (if any occurred). In the **Error Details** tab of Performance Explorer you can further drill into the errors. Double-click an error to open the TrueLog Explorer. This is only possible if you enabled **TrueLog On Error** on the **Workflow - Workload Configuration** dialog box before you started the load test.
- Click **Compare with baseline** to compare the results of this test with the results of the baseline. This button is only visible if you have already set a baseline.
- Click **Set as baseline** to make the test you have just run your baseline (your reference level) for the upcoming load tests.
- Click **Explore detailed report** to open a detailed load test report.
- Click **Open results folder** to view other result files like the virtual user report files or the virtual user output files for each virtual user.

Note: If you want to prevent the summary page to appear each time a test is complete, disable the **Show Summary Page** button in the toolbar of the **Monitor** page.

Verification Under Load

Silk Performer provides full visual verification under load capabilities for Web applications. It allows you to combine extensive content verification checks with full error drill down analysis during a load test. Silk Performer verification checks can be done from all virtual users without major performance loss.

Silk Performer's innovative TrueLog technology makes it possible to collect details regarding the errors of all virtual users that take part in a load test. Traditional probing clients, separated from the load testing tool, are only able to detect content verification errors that occur for users that actually perform verifications. Since a functional testing tool is only able to drive a limited number of concurrent users, the functional test covers only a very small subset of all users that visit your Web application. Such a probing client cannot detect verification errors that occur to users that are not contained within this small subset. Silk Performer has redefined the type of testing that can be done with load testing tools.

With Silk Performer TrueLog technology, you can find errors that usually occur to only a subset of users when your application is under a heavy load. For most applications, this is the type of load that will most likely be experienced once the application is deployed in the real world. Typical errors include incorrect text on a Web page, incorrectly computed and displayed values, or application-related messages, such as `Servlet Error` or `Server Too Busy` errors. These are not system-level errors and are displayed on Web pages with `HTTP 200` status codes.

TrueLog Explorer provides a view into Silk Performer verification-under-load capabilities with the following features:

- Visual content verification allows you to visually define the content that is to be verified.
- TrueLog On Error generation and TrueLog On Error analysis allow you to visually analyze errors to identify their root causes.

Viewing Errors in TrueLog Explorer

Before you begin, set up the option to generate TrueLog on Error files and run a test that generates errors.

As a foundation for full visual root cause analysis, Silk Performer can save the complete history of errors within a TrueLog. TrueLog On Error can even be generated in large-scale load tests. TrueLog On Error helps you uncover the true source of your application's error output under all possible load conditions.

1. When an error occurs during a load test, you can view the visual content of the TrueLog by clicking the **Explore Results** button on the workflow bar. The **Workflow - Explore Results** dialog opens.
2. Click the Silk TrueLog Explorer link. TrueLog Explorer opens with the **Step Through TrueLog** dialog box active. Select **Errors**.
3. Navigate from one occurrence of an error to the next.

   Tip: To display the history of an error, click through the preceding API nodes in the menu tree.

Performance Explorer

Performance Explorer allows you to view measures obtained through real-time monitoring and to analyze results of completed load tests. A variety of tools and features enables exhaustive analysis, reporting, and processing of all captured data.

Cloud-Based Region Summary Reports

Region summary reports gather cloud-based load test measures across geographic regions. When load tests use cloud-based agents, Silk Performer captures region-specific data for each cloud region (and for project agents, if used). Region summary reports are similar to baseline reports. Region summary report files carry the file extension `.rsr` and are stored in the results directory.

To view region summary result nodes, expand the **Region Summary Reports** node on the **Results** tab following a load test.

An overview report (similar to a baseline report) can be created for any individual region by selecting a region node on the Performance Explorer **Explore** tab and clicking **Overview Report**.

Assigning Overview Report Templates to Projects

You can specify an overview report template for each Silk Performer project. All automatically generated HTML overview reports will use this template and the template will be preselected when you create HTML overview reports manually.

1. In Silk Performer, expand the **Profiles** node in the menu tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Alternative: Choose **Settings** > **Active Profile**.

   The **Profile - [<profile name>]** dialog box opens.
3. In the shortcut list on the left, click the **Results** icon.
4. Click the **Time Series** tab.
5. Browse for a template in the **Overview report template** section and click **Open**.
6. Click **OK**.

Note: Performance Explorer's command line interface also offers the `/OVT:<template>` command for assigning overview report templates to Silk Performer projects.

Reports

Virtual User Report Files

Report files contain the simulation results for each virtual user. Details of the measurements for each individual user are presented in tabular form. The many sections in these files include reports on transactions, timers, Web page timers, and counters, and also sections on IIOP, Web form, TUXEDO, and SQL results.

Virtual User Output Files

Output files contain the output of write statements used in the test script. An output file is generated for a particular user only if write statements are executed by that user.

However, generating output files alters the time measurements of a load test. Therefore, these files should be used for debugging purposes only, and should never be generated for a full load test.

Performance Explorer Reports

Overview Report

The overview report comprises the following sections:

- General information
- Summary tables
- User types
- Custom charts
- Custom tables
- Detailed charts

General information

The general information section includes administrative information in tabular form as well as important load test results in a graphical form.

Administrative information includes the project name, a description of the project, the load test number, a description of the load test, the date of the load test, the duration of the load test, the number of used agent computers, and the number of virtual users that were running.

The charts display the number of active virtual users, response time measurements for transactions, and the number of errors that occur over time. Transaction response times are provided for successfully executed transactions, for failed transactions, and for cancelled transactions.

Additional charts display summary measurements related to the type of load testing project. For example, in the case of Web application testing, response time measurements for Web pages are presented in a graph.

Summary tables

This section contains summary measurements in tabular form, that is, aggregate measurements for all virtual users. The first table provides general information, such as the number of transactions that were executed and the number of errors that occurred. All the following tables provide summary information relevant to the type of application that was tested.

User types

For each user group, this section provides detailed measurements in tabular form. The measurements include transaction response times, individual timers, counters, and response time and throughput measurements related to the type of application that was tested (Web, database, CORBA, or TUXEDO). In addition, errors and warnings for all user groups are listed.

Custom charts

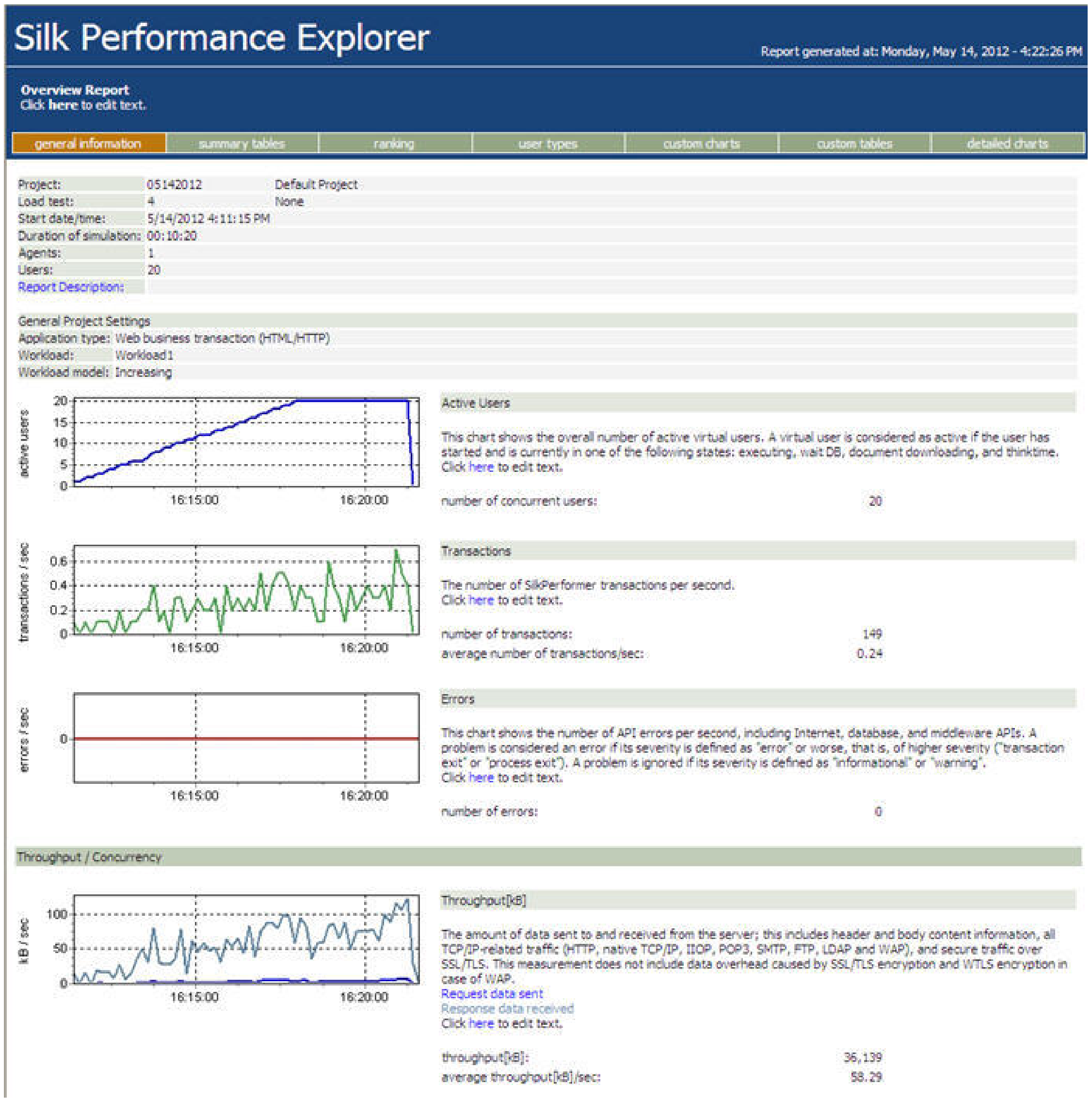
This section contains graphs that you have added manually. You can add charts to and remove charts from this section at any time. You can save your changes as a template to be displayed for every summary report.

Custom tables

This section contains tables that you have added manually. You can add tables to and remove tables from this section at any time. You can save your changes as a template to be displayed for every summary report.

Detailed charts

This section provides enlarged versions of the charts included in the report. Click a reduced version of a chart to jump to the enlarged version, and vice versa.



Project:	05142012	Default Project
Load test:	4	None
Start date/time:	5/14/2012 4:11:15 PM	
Duration of simulation:	00:10:20	
Agents:	1	
Users:	20	
Report Description:		

General Project Settings

Application type:	Web business transaction (HTML/HTTP)
Workload:	Workload1
Workload model:	Increasing

Active Users

This chart shows the overall number of active virtual users. A virtual user is considered as active if the user has started and is currently in one of the following states: executing, wait DB, document downloading, and thinktime.
Click here to edit text.

number of concurrent users: 20

Transactions

The number of SilkPerformer transactions per second.
Click here to edit text.

number of transactions: 149
average number of transactions/sec: 0.24

Errors

This chart shows the number of API errors per second, including Internet, database, and middleware APIs. A problem is considered an error if its severity is defined as "error" or worse, that is, of higher severity ("transaction exit" or "process exit"). A problem is ignored if its severity is defined as "informational" or "warning".
Click here to edit text.

number of errors: 0

Throughput[kB]

The amount of data sent to and received from the server; this includes header and body content information, all TCP/IP-related traffic (HTTP, native TCP/IP, IIOP, POP3, SMTP, FTP, LDAP and WAP), and secure traffic over SSL/TLS. This measurement does not include data overhead caused by SSL/TLS encryption and WTLS encryption in case of WAP.
Request data sent
Response data received
Click here to edit text.

throughput[kB]: 36,139
average throughput[kB]/sec: 58.29

Built-In Measures

A measure is the smallest grouping entity for the values that are collected during a load test or during a monitoring session for a particular measuring point. A measure has a name and type. A set of measured values that are related to the measure is stored as a series in a time series data (.tsd) file.

Measures can be either timers or counters. Timers collect data related to response times, counters collect data on throughput and load test events.

Performance Explorer provides a big amount of measures, which are divided into groups.

Performance Trend Reports in Silk Central

Using Silk Central for load test regression-testing allows you to track performance trends across your product builds. Monitor the evolution of transaction response times, page times, and custom measures across builds to determine how the performance of your application under test is coming along. As new features are being developed in your product, you can react in a timely manner if the application's performance behaves unexpectedly.

The following performance trend reports are available in Silk Central:

Average Page-Time Trend Report	Shows the page times per page for all tests executed for the specified test definition within the specified time range.
Average Transaction Busy-Time Trend Report	Shows the transaction busy time per transaction for all tests executed for the specified test definition within the specified time range.
Custom Measure Trend Report	Shows the average, minimum, and maximum values of the defined measure or measures for all tests executed for the specified test definition within the specified time range.
Overall Page-Time Trend Report	Shows overall page times, aggregated over all user types, for all tests executed for the specified test definition within the specified time range.
Overall Transaction Busy-Time Trend Report	Shows overall transaction busy-time, aggregated over all user types, for all tests executed for the specified test definition within the specified time range.

Refer to Silk Central Help for detailed information.

Raw Measure Data Capturing

When to use raw measure data capturing

After you have run a load test you can analyze the results in detailed reports in Performance Explorer. If you need data that is even more detailed or if you want to process collected data in another program, you can use Silk Performer to capture raw measure data and store it in CSV files. You can then, for example, import the raw measure data into a spreadsheet program and generate custom graphs.

How does Silk Performer capture raw measure data?

Silk Performer captures raw measure data in three steps:

1. Raw measure data is collected in `u@...` files (on the agents).
2. Raw measure data is merged from the `u@...` files into `i@...` files (on the agents).
3. Raw measure data is merged from the `i@...` files into `m@...` files (on the controller).

When you start a Silk Performer test, the Silk Performer controller deploys the test to the agents you have specified. Each agent starts several `perfRun.exe` processes, and within each process, several virtual users execute the test. During execution, Silk Performer creates one `u@...` file per process.

The raw measure data is then merged into `i@...` files and the `u@...` files are removed. Each `i@...` file holds the raw measure data of one measure name/measure type combination. The files are stored in the `RawData` folder: `Silk Performer <version number>/Projects/MyProject/RawData`.

Once the `i@...` files have been transferred to the controller, the raw measure data is merged into `m@...` files (also in the `RawData` folder). The `i@...` files remain in the `RawData` folder. Each `m@...` file holds the raw measure data of one name/type combination of all agents.

Step one and two of the raw measure data capturing process are executed on the agents, step three is executed on the controller.

What do the file names mean?

These are the full names of the raw measure data files:

- `u@rawdata_processid.csv`
- `i@name_type@agent.csv`
- `m@name_type@controller.csv`

The data is stored in CSV files (comma-separated value files), which allows you to easily process the raw measure data with a spreadsheet program. `u` stands for user file, `i` for intermediate file, and `m` for merged file. Here is an example of some `i@...` and `m@...` files:

```
i@ShopIt+-+Greetings_131@testingserver.csv
i@ShopIt+-+Greetings_132@testingserver.csv
i@SilkPerformer+Test+Site_131@testingserver.csv
i@SilkPerformer+Test+Site_132@testingserver.csv

m@ShopIt+-+Greetings_131@pc-johndoe.csv
m@ShopIt+-+Greetings_132@pc-johndoe.csv
m@SilkPerformer+Test+Site_131@pc-johndoe.csv
m@SilkPerformer+Test+Site_132@pc-johndoe.csv
```

Note the following from this example:

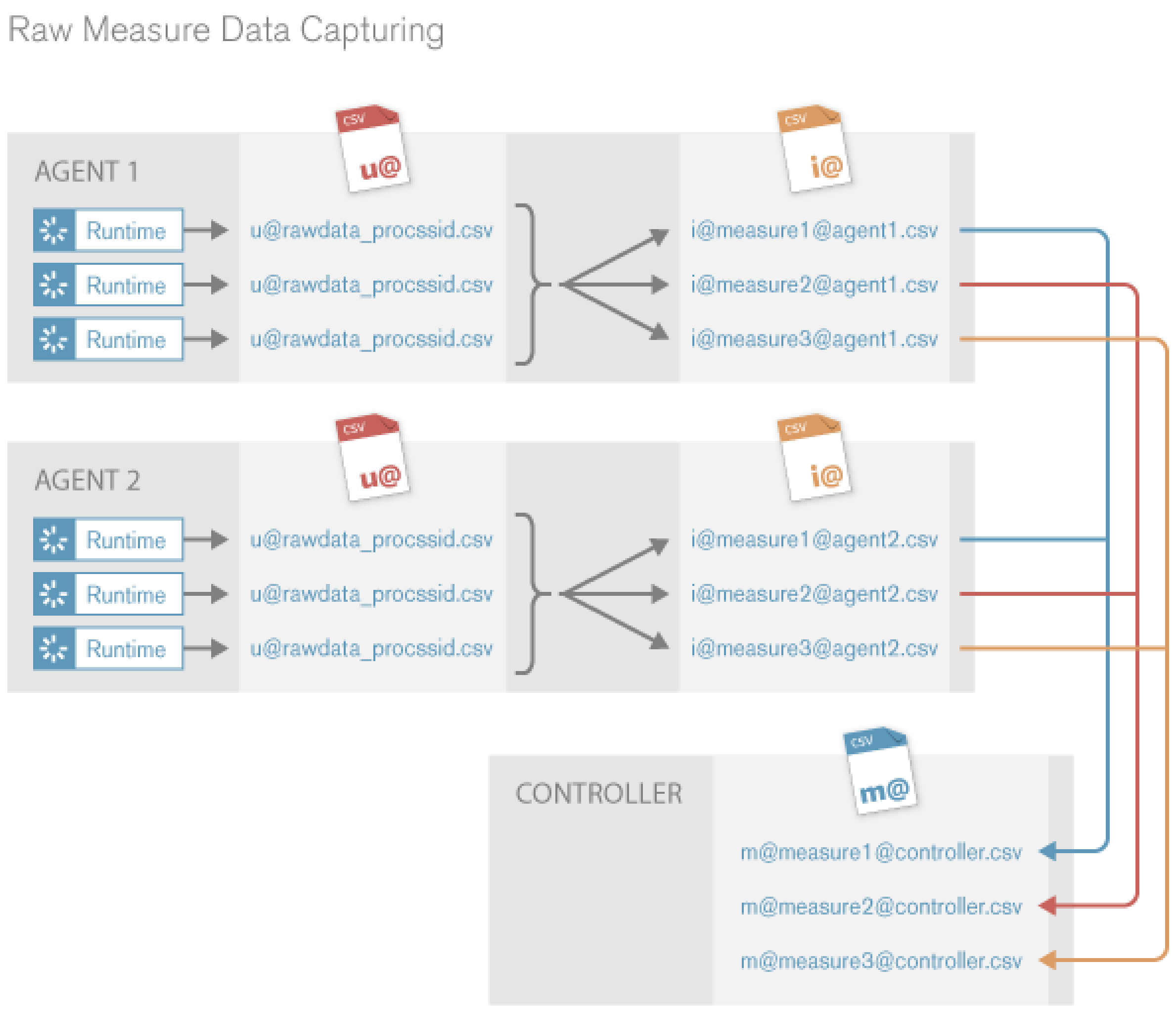
- There are no `u@...` files in this example, since they are removed when their content is merged into the `i@...` files. They are just temporary files.
- The measure names are URL-encoded (`ShopIt+-+Greetings`).
- The measure type is represented by its number code. `131` is the number code for `Page time[s]`, `132` is the number code for `Document download time[s]`.
- `testingserver` is the machine name of the agent.
- `pc-johndoe` is the machine name of the Controller.

What does a raw measure data file look like?

Here is an example of what the raw measure data in an `m@...` file looks like:

```
;MeasureName:;ShopIt - Greetings
;MeasureType:;Page time[s]
C;Timestamp;Value;VUName
;1381231408198;46;VUser
;1381231431291;31;VUser
;1381231433032;31;VUser
;1381231434120;16;VUser
;1381231436307;15;VUser
...
```

The data within the CSV files is sorted by the timestamp (which is a Unix timestamp). The timestamp represents the elapsed seconds since 01.01.1970. Time measure values are logged in milliseconds, file size values are logged in kilobytes.



Enabling Raw Measure Data Capturing

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Results** icon.
4. Click the **Measures** tab.
5. Check the **Collect raw measure data** check box.

To configure raw measure data capturing for particular measure types or single measures, use the BDL function `MeasureCollectRawData`.

Raw Measure Data File Size Statistics

The following tables provide some examples of how big raw measure data files can become when you execute load tests with the shown settings.

Note: The size values are approximate values.

Measurements	Duration of load test	Virtual users	Size of raw measure data file
1 measurement per second	1 hour	100	10 MB
3 measurements per second	5 hours	100	150 MB
1 measurement every 10 seconds	3 hours	100	3 MB

Load Test	Duration of load test	Virtual users	Size of all raw measure data files	Agents	Result packaging and download time*
10 pages (with 5 measurements per page), 1 page per second	1 hour	100	175 MB	2	30s
20 pages (with 5 measurements per page), 1 page every 3 seconds	5 hours	100	290 MB	2	45s
20 pages (with 5 measurements per page), 1 page every 3 seconds	10 hours	400	2,25 GB	4	3m 50s
20 pages (with 5 measurements per page), 1 page every 3 seconds	10 hours	4000	23 GB	40	7m 30s

* To gather these download times, Silk Performer CloudBurst agents and a 100 Mbps internet connection to the Silk Performer controller were used. The times include data processing and compression on the agents and the transfer to the controller.

Note: If you use cloud agents for your load tests, transferring large raw measure data files from the cloud agents to the controller can take a considerable amount of time.

Note: For most load test scenarios it is sufficient to generate raw measure data for only a selected set of measures. To do so, use the BDL function `MeasureCollectRawData`.

Load Testing Specific Application Types

Silk Performer supports testing within a variety of computing environments. This section explains Silk Performer's support of AJAX, SAP eCATT, terminal emulation, GUI-level testing, and much more.

Flex/AMF3 Support

This section explains Silk Performer support for the testing of Flex/AMF3 applications, including a technology overview, project setup, and Java configuration issues.

Flex/AMF3 Overview

AMF3 (Action Message Format, version 3) is a binary protocol developed by Adobe Systems. It is primarily used for remote procedure calls by Flex applications.

A primary goal in the design of AMF3 is that transmitted data be as small as possible. This requirement presents a challenge for protocol customization with Silk Performer. To make the AMF3 protocol customizable and human-readable, Silk Performer transforms AMF3 traffic into XML during script recording, and transforms the XML back into AMF3 format during replay. This means that you do not have to read binary AMF3 data in hex string format. Rather, in recorded scripts and TrueLogs, you see only XML-formatted data.

AMF0 and AMF3 support are tightly integrated (in fact AMF3 uses AMF0's message structure). Each AMF3 message begins as an AMF0 message (even when they contain the version "3" in the first two bytes of the data stream). A special AMF0 data type marker indicates where AMF0 encoding ends and AMF3 encoding begins.

Understanding Flex

According to *Adobe Systems*,

> "Flex is a free, open source framework for building highly interactive, expressive web applications that deploy consistently on all major browsers, desktops and operating systems. It provides a modern, standards-based language and programming model that supports common design patterns. MXML, a declarative XML-based language, is used to describe UI layout and behaviors, and ActionScript 3, a powerful OO programming language is used to create client logic. Flex also includes a rich component library with over 100 proven, extensible UI components for creating RIAs, as well as an interactive Flex application debugger.
>
> Rich Internet applications created with Flex can run in the browser using the ubiquitous Adobe Flash® Player software or on the desktop on Adobe AIR™. This enables Flex applications to run consistently across all major browsers and across operating systems on the desktop. And using Adobe AIR, the cross-operating system runtime, Flex applications can now access local data and system resources on the desktop.
>
> You can accelerate application development with Adobe Flex Builder 3, a highly productive, Eclipse™ based development environment, and Adobe Live Cycle Data Services ES, a set of advanced data services that can be used in Flex development. Both of these products are available for purchase."

Understanding Advanced Flex Data Services

According to Adobe ,

> "Adobe LiveCycle™ Data Services ES provides a comprehensive set of data-enabling features for using data in RIAs. It enables RIAs to talk to

back-end data and business logic in a faster, more efficient operating model. LiveCycle Data Services ES also enables seamless integration with LiveCycle ES business processes and document services.

BlazeDS is a free, open source project providing Flex Remoting and Messaging to all developers. Flex Remoting provides a binary, serialized data transport format called the ActionScript Message Format (AMF) to provide a fast, efficient means of transporting data to your RIA which accelerates application performance. Flex Remoting also makes it fast and easy for developers to connect to back-end business logic and data. Flex Messaging adds realtime data push and publish/subscribe, both powerful capabilities now made easy. Using BlazeDS, you can start using these powerful Java server integration features for free, then subscribe to the LiveCycle Data Services Community Edition for certified builds and support, or upgrade to the full LiveCycle Data Services ES Enterprise edition for a complete server solution."

Flex/AMF3 Project Setup and Testing

Prerequisites

For testing Flex/AMF3 applications based on the Adobe or BlazeDS implementation, Java Development Kit 1.5 or later is required.

For testing Flex/AMF3 applications based on the GraniteDS implementation, Java Development Kit 1.6 or later is required.

Transformation is enabled for HTTP requests and responses that have the HTTP header `Content-Type` set to `"application/x-amf"`. If you need to transform data with a different HTTP Content-Type header, see *Transformation of Custom Content-Types*.

Setting Up an Adobe Flex/AMF3 Project

For testing Flex/AMF3 applications based on the Adobe or BlazeDS implementation, select the `Flex/AMF3 (Adobe)` application type on the **Outline Project** dialog.

The `Flex/AMF3 (Adobe)` application type supports both AMF0 and AMF3. By selecting the `Flex/AMF3 (Adobe)` application type, several profile settings are automatically configured for you on the tab **Profile** > **Web (Protocol Level)** > **Transformation**:

- The `Flex/AMF3 (Adobe)` transformation DLL is selected in the **Type** drop list.
- `Transform HTTP Requests` is enabled
- `Transform HTTP Responses` is enabled
- `Enable Java Virtual Machine usage` is enabled. This setting is required because Flex/AMF3 requires a running JVM for accurate transformation of externalizable traits. This setting also causes transformed AMF3 traffic to appear in a readable XML representation that includes fewer AMF3 syntactical tags.

Setting Up a GraniteDS Flex/AMF3 Project

For testing Flex/AMF3 applications based on the GraniteDS implementation, select the `Flex/AMF3 (GraniteDS)` application type on the **Outline Project** dialog.

The `Flex/AMF3 (GraniteDS)` application type supports both AMF0 and AMF3. By selecting the `Flex/AMF3 (GraniteDS)` application type, several profile settings are automatically configured for you on the tab **Profile** > **Web (Protocol Level)** > **Transformation**:

- The `Flex/AMF3 (GraniteDS)` transformation DLL is selected in the **Type** drop list.
- `Transform HTTP Requests` is enabled
- `Transform HTTP Responses` is enabled
- `Enable Java Virtual Machine usage` is enabled. This setting is required because Flex/AMF3 requires a running JVM for accurate transformation of externalizable traits. This setting also causes transformed AMF3 traffic to appear in a readable XML representation that includes fewer AMF3 syntactical tags.

Configuring a Customized GraniteDS Configuration

If your application under test uses a customized GraniteDS configuration, you may find that an exception is reported in TrueLog Explorer, in the Recorder log or the virtual user log files. These exceptions resemble the following:

```
WebPagePost(WebEngine: 82 - Content transformation error, RESPONSE: AMF3:
Java based XML generation failed,
using fallback, reason:
org.granite.messaging.amf.io.AMF3SerializationException
…
Caused by: java.lang.RuntimeException: The ActionScript3 class bound to
org.granite.example.addressbook.entity.Person
(ie: [RemoteClass(alias="org.granite.example.addressbook.entity.Person")])
implements flash.utils.IExternalizable but
this Java class neither implements java.io.Externalizable nor is in the scope
of a configured externalizer
(please fix your granite-config.xml)
…
```

To make Silk Performer aware of this custom configuration and to get rid of the exceptions, proceed as follows:

1. Locate the customized configurations of your application under test. These can usually be found in `granite-config.xml` or `granite-custom-config.xml`, which is located in the application's classpath.
2. In the **Project** menu tree, locate the `granite-custom-config.xml` file in the **Data Files** folder.
3. Double-click `granite-custom-config.xml` to open it in the editor.
4. Copy and paste the customized configurations that you located in step 1 into the `granite-custom-config.xml` data file and save your changes.

Modeling a Script

Using the JVM requires the configuration of Flex/AMF3 application-specific custom JAR files that contain the classes that are necessary for serialization into correct XML representation. For this reason, when modeling a Flex/AMF3 script, the **Model Script** dialog includes an **Add Custom JAR File(s)** button. Click this button to browse to and **Add** any custom JAR files that are specific to the application under test. Added JAR files are displayed within the Project tree **Data Files** node.

These required JAR files (or individual `.class` files) are located on the server to be tested. These files must be prepared manually and copied from the server to the Silk Performer controller machine. It may be that the only required task is to place individual `.class` files into an archive.

Note: JAR files must be placed in the `Project` folder.

Clicking **Settings** on the **Model Script** dialog links you directly to the current user profile, **Java Settings** tab. Use this to change Java settings for the currently selected user profile.

The Apache Flex 2, 3, 4 and BlazeDS 3.2.0 base JAR files are installed in the directory `C:\Program Files(x86)\Silk\Silk Performer <version>\ClassFiles\Adobe-Flex`. These JAR files are automatically added to the classpath. These files may be manually updated by copying new versions of `flex-messaging-common.jar` and `flex-messaging-core.jar` into the directory.

Make sure that you have the GraniteDS base JAR files on your machine and then add these JAR files to the classpath.

Note: Do not enable **Use system classpath** on the **Java Settings** tab. JAR files set in the system classpath may overrule manually configured JAR files.

Generating a Script from a Capture File

While recording a user transaction, the Silk Performer recorder creates a so-called Silk Performer capture file, which contains the entire traffic of the recorded session. After saving, the capture file is opened in the Workbench for further analysis and processing. Before generating a script from the captured traffic, you can configure recording rules and other settings, which are applied during the script generation process.

If any errors occur, click the buttons in the **Resolve Problems** area to resolve them. Then, click **Generate Script** to generate a script from the capture file.

On 64-bit operating systems, both a 32-bit and 64-bit Java installation are required. The 32-bit installation is used for replaying scripts, the 64-bit installation is used for generating scripts. If a 64-bit installation is not available, you can force Silk Performer to use a 32-bit process for script generation by setting the following registry key to 1: `HKEY_LOCAL_MACHINE\SOFTWARE\Silk\SilkPerformer\<version>\Force32BitCaptureAnalyzer`.

Attention: Using the 32-bit script generator can cause issues with large capture files.

Replaying a Script

If your Java configurations are incorrect (for example, if JAR files are missing), XML responses (only visible in TrueLog Explorer) will not be generated in an easily readable format. Also warnings or errors will be written to the Virtual User Output pane. Typically, errors and warnings indicate whether or not they were caused by requests (client to server) or responses (server to client).

Customizing AMF3 Scripts

Do not change the overall structure of XML objects within AMF3 scripts. It is okay to parse values or insert verification functions, but deleting or rearranging AMF3 XML elements will destroy a call. Also do not change the order of elements within arrays.

Code Examples

Command message example:

```
<AmfXml
version="3">
  <Msg length="203" operation="null" responseURI="/
1">
    <StrictArray
nrElems="1">

<Amf3>

<JavaObject>

<flex.messaging.messages.CommandMessage>
            <destination></
destination>
            <messageId>3EE7E87A-96E6-3272-30F7-FFEF6B9EFE9E</
messageId>
            <timestamp>0</
timestamp>
```

```
            <timeToLive>0</
timeToLive>

<headers>

<entry>
                <string>DSId</
string>
                <string>nil</
string>
              </
entry>
            </
headers>
            <body class="flex.messaging.io.amf.ASObject"
serialization="custom">
              <unserializable-parents></unserializable-
parents>

<map>

<default>
                  <loadFactor>0.75</
loadFactor>
                  <threshold>12</
threshold>
                </
default>
                <int>16</
int>
                <int>0</
int>
              </
map>

<flex.messaging.io.amf.ASObject>

<default>
                  <inHashCode>false</
inHashCode>
                  <inToString>false</
inToString>
                </
default>
              </
flex.messaging.io.amf.ASObject>
            </
body>
            <correlationId></
correlationId>
            <operation>5</
operation>
          </
flex.messaging.messages.CommandMessage>
        </
JavaObject>
      </
Amf3>
    </
StrictArray>
  </
Msg>
</AmfXml>
```

Remoting message example:

```
<?xml version='1.0' encoding='UTF-8'?
>
<AmfXml
version="3">
  <Msg length="394" operation="null" responseURI="/
4">
    <StrictArray
nrElems="1">

<Amf3>

<JavaObject>

<flex.messaging.messages.RemotingMessage>
            <clientId class="string">208F18FC-6F0D-9964-3746-76438D73A6A3</
clientId>
            <destination>myextservice</
destination>
            <messageId>A98651D4-690D-E530-B744-FFEF866937C7</
messageId>
            <timestamp>0</
timestamp>
            <timeToLive>0</
timeToLive>

<headers>

<entry>
                <string>DSId</
string>
                <string>208F0A62-A409-8994-08AF-223CD381602C</
string>
              </
entry>

<entry>
                <string>DSEndpoint</
string>
                <string>my-amf</
string>
              </
entry>
            </
headers>
            <operation>setMyExtClass</operation>

<parameters>

<com.borland.silkperformer.flex.samples.MyExt>
                <string>string1</
string>
                <string>string2</
string>
                <string>string3</
string>
                <boolean>true</
boolean>
                <date>2009-03-13 14:05:01.545 CET</
date>
              </
com.borland.silkperformer.flex.samples.MyExt>
            </
```

```
parameters>
          </
flex.messaging.messages.RemotingMessage>
        </
JavaObject>
      </
Amf3>
    </
StrictArray>
  </
Msg>

</AmfXml>
```

Acknowledge message example:

```
<?xml version='1.0' encoding='UTF-8'?
>
<AmfXml
version="3">
  <Msg operation="/5/onResult"
responseURI="">

<Amf3>

<JavaObject>

<flex.messaging.messages.AcknowledgeMessage>
          <clientId class="string">208F18FC-6F0D-9964-3746-76438D73A6A3</
clientId>
          <messageId>20903C27-020C-F5E6-68E7-C8175449798D</
messageId>
          <timestamp>1236954453026</
timestamp>
          <timeToLive>0</
timeToLive>
          <body
class="com.borland.silkperformer.flex.samples.MyExt">
            <string>some string</
string>

<com.borland.silkperformer.flex.samples.MyExt>
              <string>string1</
string>
              <string>string2</
string>
              <string>string3</
string>
              <boolean>true</
boolean>
              <date>2009-03-13 14:04:58.14 CET</
date>
            </
com.borland.silkperformer.flex.samples.MyExt>
            <string>GET</
string>
            <string>string</
string>
            <boolean>true</
boolean>
            <date>2009-03-13 14:05:01.545 CET</
date>
          </
body>
```

```
          <correlationId>C4BD7297-67CB-84AB-0C1A-FFEF8DEB3A97</
correlationId>
        </
flex.messaging.messages.AcknowledgeMessage>
      </
JavaObject>
    </
Amf3>
  </
Msg>
</AmfXml>
```

Understanding Flex/AMF3 Scripts

This section explains aspects of the Flex/AMF3 protocol including script elements and XML Representation of Binary AMF.

Flex/AMF Packet-Oriented Protocol

Flex/AMF uses a binary packet-oriented protocol that is passed with `HTTP POST` requests. This packet format is called Action Message Format (AMF). The Flex/AMF application sends AMF packets to the server in the bodies of `HTTP POST` requests. The server responds with AMF packets in the bodies of HTTP responses. Both the HTTP requests and the responses contain the `Content-Type` HTTP header with the value `application/x-amf`.

HTTP requests that carry AMF packets are recorded with the function `WebPagePost`. The URL to be used is recorded as the first parameter of the function, the AMF packet to be sent is recorded as the second parameter.

In scripts, logs, and TrueLogs, Silk Performer does not display binary AMF packets in their original binary format. Instead, an XML based textual representation is used. The conversion from this textual representation to binary AMF and vice versa is done transparently by the recorder and replay. Binary AMF can only be seen in the textual log if the appropriate logging options are enabled.

The advantage of this is that AMF packets are easy to read, understand and customize. AMF packets are also rendered using the XML tree view in TrueLog Explorer, and can be customized there in the same way that all XML data can be customized.

XML Representation of Binary AMF

The root node of Silk Performer XML representation of binary AMF is called `AmfXml`. Its only attribute is the `version` attribute, which corresponds to the `version` field in binary AMF and denotes the AMF protocol specification version. Its value is always `zero (0)`.

AMF packets consist of a list of context headers and a list of messages. Any of these lists may contain zero elements.

The basic structure of an AMF packet in XML is shown below:

```
<?xml version='1.0' encoding='UTF-8'?>
<AmfXml version=\"0\">
<CtxHeader length="112" mustUnderstand="true"  name="some_name">
     (contents of context header)
  </CtxHeader>
    (more context headers)
  <Msg length="5" operation="some.operation" responseURI="/1">
     (contents of message)
  </Msg>
    (more messages)
</AmfXml>
```

Context Headers

Context headers have the attributes `length`, `mustUnderstand`, and `name`.

The value of the `length` attribute specifies the number of bytes used for this context header in the binary AMF packet. It is for informational purposes only, and need not be adjusted if the XML representation is customized because the transformation from XML to binary AMF automatically calculates the correct value.

The content of a context header is exactly one typed value.

Messages

Messages have the attributes `length`, `operation`, and `responseURI`. The value of the `length` attribute specifies the number of bytes used for this message in the binary AMF packet. It is for informational purposes only, and need not be adjusted if the XML representation is customized because the transformation from XML to binary AMF automatically calculates the correct value.

The content of a message is exactly one typed value.

Typed Values

Typed values are serialized actionscript objects. Each context header and each message contains exactly one typed value.

Typed values can be arbitrary complex, hierarchical data structures.

Each typed value consists of a `type`, a `value`, and optionally a `name`.

The type is specified by the XML node name. The optional name is specified by the `name` attribute of the XML node.

For simple types, the `value` is specified in the content of the XML node. For container types, the `value` is specified by subnodes of the XML node.

Some types may have additional attributes.

Typed values have a `name` attribute only when they are direct subnodes of an `Object`, `TypedObject`, or `ECMAArray`. Top-level typed objects and subnodes of a `StrictArray` do not have name attributes.

The following table offers an overview of available types and optional attributes.

XML Nodename	**Value location**	**Additional attributes(except name, see above)**
Number	Content of node	---
Boolean	Content of node	---
String	Content of node	---
Object	Subnodes	refId (optional)
Null	no value	---
Undefined	no value	---
Reference	Content of node	---
ECMAArray	Subnodes	nrElems, refId (optional)
StrictArray	Subnodes	nrElems, refId (optional)
Date	Content of node	---
LongString	Content of node	---
Unsupported	no value	---
XMLObject	Content of node	refId (optional)
TypedObject	Subnodes	type, refId (optional)

The optional `refId` attribute of the types `Object`, `ECMAArray`, `StrictArray`, `XMLObject`, and `TypedObject` can be used to assign an arbitrary ID to such a value. Such an ID can then be used to reference the value later with a `Reference` node.

This technique is commonly used to avoid multiple identical representations of the same object. With this approach, an object is represented only once, and later it is only referenced by a `Reference` node. This allows you to efficiently serialize complex data structures, which may even contain cyclic references.

Examples of Typed Values

```
<Object>
  <Boolean name="coldfusion">true</Boolean>
  <Boolean name="amfheaders">false</Boolean>
  <Boolean name="amf">false</Boolean>
  <Boolean name="httpheaders">false</Boolean>
  <Boolean name="recordset">true</Boolean>
  <Boolean name="error">true</Boolean>
  <Boolean name="trace">true</Boolean>
  <Boolean name="m_debug">true</Boolean>
</Object>

<ECMAArray nrElems="0">
  <Number name="Time">1080677591838</Number>
  <String name="EventType">Information</String>
  <Date name="Date">2004-03-30 20:13:11.838 TZ:-480 [426F73ACED63C000FE20]</
Date>
  <String name="Message">CF_DEBUG_DISABLED</String>
  <String name="Source">Server</String>
</ECMAArray>
```

Example request sent to sample application

```
<?xml version='1.0' encoding='UTF-8'?>
<AmfXml version="0">
  <CtxHeader length="82" mustUnderstand="false" name="playerInfo">
    <Object>
      <String name="version">WIN 7,0,19,0</String>
      <String name="manufacturer">Macromedia Windows</String>
      <String name="os">Windows 2000</String>
    </Object>
  </CtxHeader>
  <Msg length="13" operation="petmarket.api.catalogservice.getCategories"
       responseURI="/4">
    <StrictArray nrElems="1">
      <String>en_US</String>
    </StrictArray>
  </Msg>
</AmfXml>
```

Example request received from sample application

```
<?xml version='1.0' encoding='UTF-8'?>
<AmfXml version="0">
  <Msg operation="/4/onResult" responseURI="null">
    <TypedObject type="RecordSet">
      <ECMAArray name="serverinfo" nrElems="0">
        <StrictArray name="initialData" nrElems="5">
          <StrictArray nrElems="4">
            <Number>5</Number>
            <String>birds</String>
            <String>birds</String>
            <String>D7A91E</String>
          </StrictArray>
          <StrictArray nrElems="4">
```

```
            <Number>4</Number>
            <String>cats</String>
            <String>cats</String>
            <String>FFA672</String>
          </StrictArray>
          <StrictArray nrElems="4">
            <Number>2</Number>
            <String>dogs</String>
            <String>dogs</String>
            <String>FF876F</String>
          </StrictArray>
          <StrictArray nrElems="4">
            <Number>1</Number>
            <String>fish</String>
            <String>fish</String>
            <String>4F9FDB</String>
          </StrictArray>
          <StrictArray nrElems="4">
            <Number>3</Number>
            <String>reptiles</String>
            <String>reptiles</String>
            <String>97D76B</String>
          </StrictArray>
        </StrictArray>
        <Number name="version">1</Number>
        <Number name="totalCount">5</Number>
        <String name="serviceName">PageableResultSet</String>
        <Number name="cursor">1</Number>
        <StrictArray name="columnNames" nrElems="4">
          <String>CATEGORYOID</String>
          <String>CATEGORYDISPLAYNAME</String>
          <String>CATEGORYNAME</String>
          <String>COLOR</String>
        </StrictArray>
        <Null name="id"></Null>
      </ECMAArray>
    </TypedObject>
  </Msg>
</AmfXml>
```

Date Values

Binary AMF represents date values with a 64-bit floating point value that specifies the number of milliseconds elapsed since January 1, 1970, midnight GMT, and a 16-bit signed integer value that specifies the timezone offset in minutes relative to GMT.

Date values are represented in XML in the following format: `YYYY-MM-DD HH:MM:SS.mmm TZ:[+-]tzoffset`

YYYY	denotes a 4 digit year
MM	denotes a 2 digit month (range: 01-12)
DD	denotes a 2 digit day (range: 01-31);
HH	denotes a 2 digit hour-of-day (range: 00-23)
MM	denotes a 2 digit minute value (range: 00-59)
SS	denotes a 2 digit seconds value (range: 00-59)
mmm	denotes the 3 digit number of milliseconds value
tzoffset	denotes the timezone offset to GMT in minutes

Because of the rounding of floating point values to the nearest millisecond it is generally not possible to reconstruct an exact binary representation from these string representations. For this reason, the original binary representation is also included in hexadecimal representation in square brackets. As long as such a date value is not customized, the hexadecimal representation can be used to exactly reconstruct the original date value when XML is transformed back to binary AMF.

When customizing a date value, the hexadecimal representation can either be deleted or left untouched. Regardless it is ignored when the XML representation is converted to binary AMF.

Example string representation of a date value: `2004-03-30·16:38:38.260·TZ:-300·[426F73945F768000FED4]`

Out-dated AMF3 Data Types

Simple Data Types

Note: This topic is included for backward compatibility purposes only. It describes an out-dated method of XML serialization. If your Java settings are correctly configured and you are using Silk Performer version 2009 or later, you may safely ignore this topic. Usage of the JVM requires the configuration of Flex/AMF3 application-specific custom JAR files that contain classes that are necessary for the serialization of code into correct XML representation. When these JAR files are missing, the translation DLL uses the method of XML serialization described in this topic.

Here are four simple data types that are provided by AMF3. In the binary protocol, each of these data types is transmitted as a 1-byte type marker. The corresponding XML tags for each data type are included in parenthesis.

- Undefined (`<Undefined></Undefined>`)
- Null (`<Null></Null>`)
- False (`<False></False>`)
- True (`<True></True>`)

Integer

Integers in AMF3 are serialized as variable-length 29-bit unsigned integers. The variable-length format ensures that integers take up as little space as necessary. In XML format, integers look like this:

```
<Integer>123456</Integer>
```

The integer data type is encoded as a 1-byte type marker, followed by up to 4 bytes that encode the integer using the variable length format. Besides their use as a data type, variable-length integers are used in other places within the protocol.

Double

A double stores an 8-byte double precision floating point value as defined in IEEE 754. This is the data type that many programming languages refer to as a *double*. In XML format, doubles look like this:

```
<Double>3.14159</Double>
```

In the binary format, doubles are encoded as 1-byte type markers followed by 8 bytes that contain the double value.

String

Strings are used to store all types of UTF-8-encoded string data. Besides their use as a data type, strings are also used to specify string data, such as classes or member names. All string values in an AMF3 data stream are put into a reference table which is typically generated when AMF3 or XML data is parsed.

A string can either be encoded as a string value or as a string reference. A string reference refers to a previous occurrence of its number in the string reference table. This feature is used to save space when the same string value occurs several times in a single AMF3 data stream.

In XML format, a string looks like this:

```
<String>Hello, World!</String>
```

When string references are used, the referenced string is marked with a reference ID (in the form of an attribute), and the string reference refers to the reference ID:

```
<String val-refId="#0">This string is referenced</String>
<String val-ref="#0"></String>
```

XML and XMLDocument

AMF3 provides two data types to encode XML data: `XMLDocument`, which represents a Flash legacy data type, and `XML`, which is a newer XML type. Both data types encode XML as strings. The difference is that these strings are not registered in the string reference table. They are registered in the object reference table.

Note: With the XML format, the actual XML content must be XML-encoded. Due to a limitation with Silk Performer's XML parser, it is currently not possible to use XML `CDATA` sections to notate such data.

```
<Xml>&lt;Data&gt;some data&lt;/Data&gt;</Xml>
<XmlDoc>&lt;Data&gt;some more data&lt;/Data&gt;</XmlDoc>
```

As with strings, XML data can be referenced to save space:

```
<Xml refId="data">&lt;Data&gt;foobar&lt;/Data&gt;</Xml>
<Xml ref="data"></Xml>
```

Date

In AMF3, the date is encoded as the number of milliseconds elapsed since January 1st 1970, 00:00, in the UTC timezone. The number of milliseconds is encoded as a double. Dates also support references, and each date is registered in the object reference table.

```
<Date>2007-02-18 11:29:00.000</Date>
<Date refId="newmillenium">2000-01-01 00:00:00.000</Date>
<Date ref="newmillenium"></Date>
```

Array

Arrays in AMF3 cover two types of data structures: *standard arrays*, where each element is identified by an index number, and *associative arrays,* where each element is identified by a key (in the case of AMF3, this is always a string). In the binary representation of an array, the associative portion of the array is listed first. This is followed by a separator (an empty string) and finally the dense portion of the array (the standard array elements).

In XML representation, the elements of an associative array are marked by their name attributes. Generally, arrays are not bound to a specific type, which means that you can put any data type in them, including different data types.

```
    <Array>
        <Integer name="number1">23</Integer>
        <Integer name="number2">42</Integer>
        <String name="username">EMEA\johndoe</String>
        <Date>2008-02-17 13:47:13.000</Date>
        <Undefined></Undefined>
    </Array>
```

In the above example there is an array with three elements that have a name attribute (meaning they belong to the associative portion of the array) and two elements that do not have a name attribute, which means that they belong to the dense portion of the array.

As with other data types, arrays can be referenced; their elements can be referenced; even their elements' names can be referenced:

```
    <Array>
        <String name="foo" name-refId="#0">bar</String>
        <String val-ref="#0"></String>
    </Array>
```

In this example, there is an array with two string elements. The first element has the name `foo` and the value `bar`. The second element has no name, but references the string with the string reference table ID `#0` (the string `foo`).

Arrays themselves are registered in the object reference table.

Object

Objects are the most powerful data type within AMF3. There are several subtypes of classes that you need to be aware of. What all object types have in common is that they bear a class name.

Object Traits

The traits of an object are its member names, plus information indicating if an object is dynamic or externalizable.

An object with traits contains a list of member names followed by the corresponding members. In binary format, member names are encoded as strings without type markers. The number of member names is encoded in an additional field of flags that also contains information about the object's sub type. Following the members, an optional, additional list of dynamic members, each preceded by their respective member name, is included. The list of dynamic members is only included if the dynamic flag is set to `true` and terminated by an empty string.

```
    <Object classname="testclass">
        <Member>strFirstName</Member>
        <Member>strSurname</Member>
        <String>John</String>
        <String>Doe</String>
    </Object>

    <Object classname="testclass2" dynamic="true">
        <Integer name="answer">42</Integer>
    </Object>
```

Object References

Objects can be directly referenced because they are registered in the objects reference table.

```
<Object classname="testclass3" dynamic="true" refId="#0">
    <Integer name="answer">42</Integer>
</Object>
<Object ref="#0"></Object>
```

Traits References

Each object of subtype *object traits* or *externalizable traits* is not only added to the objects reference table, but also to the traits reference table. Other objects can then refer to the traits of these objects. This means that such an object does not need to come with the list of members and the object flags by itself, but only with the actual members.

```
<Object classname="nameclass" traits-refId="name">
        <Member>userid</Member>
        <Member>firstName</Member>
        <Member>lastName</Member>
        <String>jdoe</String>
        <String>John</String>
        <String>Doe</String>
    </Object>
    <Object traits-ref="name">
        <String>mmustermann</String>
        <String>Max</String>
        <String>Mustermann</String>
    </Object>
```

Externalizable Traits

Externalizable traits contain the binary representation of an object, an indeterminate number of bytes serialized in an unknown format. The format depends on the class that serializes and deserializes this data. The class also has to know how many bytes it has to consume from the byte stream.

Silk Performer provides support for the three most common classes:

- `flex.messaging.io.ArrayCollection`
- `flex.messaging.io.ArrayList`
- `flex.messaging.io.ObjectProxy`

According to Adobe, these three classes are the most common, and are encoded as AMF3. This means that these three classes are parsed like normal arrays or objects, respectively.

In case a class other than these three is found, it is assumed that all bytes up to the end of the stream belong to the externalizable traits. The content of such an unknown externalizable trait is stored as Base64-encoded data.

```
<Object classname="flex.messaging.io.ArrayCollection" externalizable="true">
<Array>
<Integer>1</Integer>
<Integer>2</Integer>
<Integer>3</Integer>
<Integer>4</Integer>
</Array>
</Object>

<Object classname="flex.messaging.io.ArrayList" externalizable="true">
<Array>
<String>hugo</String>
<String>hugo2</String>
</Array>
</Object>

<Object classname="flex.messaging.io.ObjectProxy" externalizable="true">
<Object classname="" dynamic="true">
<String name="ssnum">555-55-5555</String>
<String name="name">Tyler</String>
<Integer name="age">5</Integer>
</Object>
</Object>

    <Object classname="test" externalizable="true">SGVsbG8sIHdvcmxk</Object>
```

ByteArray

A ByteArray holds an array of bytes. In XML representation, binary data is encoded in Base64 format.

ByteArrays are registered in the object reference table.

```
<ByteArray>SGVsbG8sIHdvcmxk</ByteArray>
```

References

Three different reference tables are used. The following list offers an overview of which data types are registered in which reference table.

String Reference Table:

- String
- Class names (not an actual data type)
- Member names (not an actual data type)
- Value name of elements of the associative portion of an array (not an actual data type)

Object Reference Table:

- Object
- Date
- Array
- ByteArray
- XML
- XmlDocument

Traits Reference Table:

- Object traits
- Externalizable traits

Not registered in any reference table:

- Undefined
- Null
- False
- True
- Integer
- Double

Reference IDs are generated by the AMF3 parser. However in hand-written XML documents, you can use custom reference IDs:

```
<String val-refId="username">EMEA\johndoe</String>
<String val-ref="username"></String>
```

Note: Custom reference IDs are lost when an XML document is transformed from XML to AMF3 and back to XML because the AMF3 generator simply looks up the reference table position using the reference ID string and only encodes the position in the AMF3. When parsing such a generated AMF3 data stream, the parser no longer has the information about the custom reference IDs. It generates reference IDs that begin with # and a continuous number that directly reflects the position in the string reference table.

Customizing Flex/AMF3 Scripts

Flex/AMF3 scripts are best customized using TrueLog Explorer.

TrueLog Explorer presents the XML representation of binary AMF3 requests and response bodies in a tree view structure. The HTTP request body is presented in **Request** view. The HTTP response is presented in **Response** view. The textual representation of request bodies is shown in **Out Header** view. The textual representation of response bodies is shown in **In Header** view.

As with other XML based applications, verification and parsing functions can be inserted visually using TrueLog Explorer.

Web Applications Support

Silk Performer supports testing of all types of Web applications, both on the protocol level (HTTP) as well as on a browser-driven basis.

Web Application Communication Overview

The topics in this section offer an overview of synchronous and asynchronous Web application processing flow, including the three main asynchronous communication models. They also examine the implications of AJAX on automated load testing and the use of XPATH in identifying DOM elements.

Understanding Synchronous and Asynchronous Communication

In traditional Web-based applications, a user input triggers a number of resource requests. Once the requests have been answered by the server, no further communication takes place until the user's next input. Such communication between client and server is known as *synchronous communication*.

Here is an example of traditional synchronous communication passing between a browser and a Web server:

1. The user clicks a UI control in a browser-based web application.
2. The browser converts the user's action into one or more HTTP requests and passes them along to the Web-application server.
3. The application server responds to the user's requests by returning the requested data to the user. At this point the application is updated and the synchronous communication loop is complete. A new synchronous communication loop will begin when the user next clicks a UI control in their browser.

Synchronous communication is limited due to the lapses in application updates that are presented to the user at regular intervals. Even if a synchronous application is designed so that it automatically refreshes information from the application server at regular intervals (for example, every 12 seconds), there will still be consistent periods of delay between data refreshes. For many applications, such update delays don't present an issue because the data they manage don't change often. Some application types however, for example stock-trading applications, rely on continuously updated information to provide optimum functionality and usability to their users.

Web 2.0 web-based applications address this issue by relying on *asynchronous communication.* Asynchronous applications deliver continuously updated application data to users. This is achieved by separating client requests from application updates. Multiple asynchronous communications between client and server may occur simultaneously or in parallel with one another.

While asynchronous communication delivers tremendous value to users, it presents a serious challenge to software-testing tool vendors who have difficulty emulating it with traditional test scripts.

Asynchronous Communication Models

There are three main types of asynchronous request and response sequences: *push, poll,* and *long-poll.* When building a test script for a web-based application, it is essential that you understand which type of asynchronous communication is in use.

Polling Communication Model

With this model of asynchronous communication, the client sends HTTP requests to the application server at a consistent rate, for example every 8 seconds. The server returns updates to the browser, thereby keeping the application updated with intermittent frequency.

Long-Polling Communication Model

With this model the browser sends an HTTP request to the application server. The server responds with an HTTP response whenever there is an update. The client generates an HTTP request to a known address on the server. Immediately following receipt of the server response, the browser sends out another HTTP request. It is the immediate response by the browser that differentiates this model from the standard polling communication model. The immediate response leads to longer wait time on the server side for a server response.

Push Communication Model

As with the polling and long-polling models, with the push model, communication begins with the browser sending an HTTP request to the application server. The response returned by the server however is kept open. This results in the browser keeping the connection to the server open. The server then sends a sub-message over the open connection whenever it has an update. With this model, communication between browser and server remains open indefinitely.

The server can close the connection at any time or keep it open even in the absence of new updates by sending ping messages that prevent the browser from closing the connection due to time-outs.

Testing Asynchronous Communication on the Protocol Level

The asynchronous testing functionality for protocol-level record/replay is designed for Web applications that use asynchronous communication patterns such as polling, long-polling, and push. The characteristics of such applications is periodic, event-based, or server-triggered content updates without user interaction. Asynchronous web application testing on protocol-level is less resource intensive, but it is more challenging to script as opposed to the BDLT approach, which is resource intensive but automates the scripting process entirely during recording.

Recording Asynchronous Communication

Silk Performer offers a dedicated project type (`Web (Async)`) to facilitate recording of web applications that make use of asynchronous communication patterns. While recording a user transaction, the Silk Performer recorder creates a so-called Silk Performer capture file, which contains the entire traffic of the recorded session. After saving, the capture file is opened in the Workbench for further analysis and processing. Before generating a script from the captured traffic, you can configure recording rules and other settings, which are applied during the script generation process. Note that it is possible to generate several scripts (with different options) from the same capture file.

Asynchronous Web Functions

For each of the previously mentioned asynchronous communication patterns Silk Performer offers a web API function to create an asynchronous communication channel to the web server:

- `WebAsyncPreparePush`
- `WebAsyncPreparePoll`
- `WebAsyncPrepareLongPoll`

Each `WebAsyncPrepare...` function call starts a dedicated asynchronous communication channel with the subsequent web function call.

The `WebAsyncPrepare...` functions take an optional BDL callback function as parameter. The callback function is called to notify the virtual user about certain events. For a detailed description of the events, refer to the BDL reference.

Asynchronous communication channels are active in parallel to normal virtual user activity, whereas callback functions are called by the web replay engine at many but specific occasions during script execution.

Synchronizing Callbacks with Virtual User Activity

Sometimes virtual user activity needs to be synchronized with callback function execution. A typical example is a situation where a virtual user needs to wait for information from the server, which is delivered via an asynchronous communication channel.

At the time where the information is required, the virtual user calls the function `UserWaitFor`. When the data arrives, the related callback function is called. This function retrieves and stores the awaited information. At this stage, the callback function signals an event to the virtual user by calling the function `UserSignal`. The virtual user can then safely access the information stored by the callback function.

Testing WebSocket Connections

The WebSocket protocol is a TCP-based network protocol that allows to establish a full-duplex (bidirectional) connection between client and server. A conventional HTTP connection follows the request-response principle: each client request triggers a server response.

To establish a WebSocket connection, the client sends a WebSocket upgrade request embedded in an HTTP message. Once acknowledged by the server, the open connection can be used for communication by both the server and the client at any time.

Using the WebSocket protocol results in reduced network traffic and latency. It is an alternative to communication models such as polling and long-polling that were used to simulate full-duplex connections.

You can either create a test script by manually scripting the respective functions, or you can use the Silk Performer Recorder to do the scripting for you. To learn about the WebSocket BDL functions, refer to the BDL Reference.

There are two functions you can use to establish a connection to a WebSocket server: Scripting `WebSocketConnect` establishes an asynchronous communication channel. Scripting `WebSocketConnectSync` establishes a synchronous communication channel.

Note: The Recorder will always follow the asynchronous scripting model by generating `WebSocketConnect` functions and associated callback function stubs.

The WebSocket protocol provides for sending and receiving both text and binary messages. Accordingly, Silk Performer offers `WebSocketSendTextMessage` and `WebSocketSendBinaryMessage` to support these message types.

HTTP/2 Support

General

HTTP/2 is a major revision of the hypertext transfer protocol (HTTP) used in the World Wide Web. The protocol was derived from an experimental protocol named SPDY, introduced by Google, before it was published as RFC 7540 in 2015.

When designing HTTP/2, the inventors put their focus on efficiency. For that reason, the protocol offers a series of improvements over HTTP/1.1:

- HTTP/2 uses only one connection between client and server. This single connection supports multiple communication streams in parallel, which reduces the overhead for establishing multiple connections between client and server. In contrast: An HTTP/1.1 connection only allows simple request/response patterns.
- HTTP/2 is a binary protocol. As such it is not readable as opposed to HTTP/1.x.
- HTTP/2 uses a compression method that has been specifically designed for HTTP headers. This compression method takes into account that often very similar headers get sent over and over again.
- HTTP/2 allows a server to push resources to the client without an explicit request from the client.

Configuring HTTP/2

The Silk Performer HTTP/2 support is disabled by default. You can enable and configure HTTP/2 in the Profile Settings: Click **Settings** > **Active Profile** > **Web (Protocol Level)**.

Limitations

Currently, Silk Performer supports HTTP/2 for replay only. Scripts recorded using HTTP/1.1 will replay correctly with HTTP/2 unless the server returns specific content that depends on the HTTP version the client uses.

The Silk Performer Recorder does not allow a browser to use HTTP/2, thus HTTP 1.1 is used. In most cases, this does not affect the recorded traffic and content. If, however, the server returns different content depending on the HTTP version used for communication, the HTTP/2 content cannot be recorded.

Server Push Support

HTTP/2 allows a server to push resources to the client rather than waiting for the client to parse the main HTML document and then request further resources one by one. Typically, HTTP/2 servers push files like style sheets and images to the client. Clients usually require these resources anyway to render a page.

You can enable and disable server push in the Profile Settings: Click **Settings** > **Active Profile** > **Web (Protocol Level)**.

- In TrueLog Explorer pushed resources are represented by nodes with **(server push)** appended. For these nodes the tabs **Out Header** and **Out Body** are disabled. If the resource is required in subsequent pages, cache hits are generated as usual.
- If you request a page several times, the server will attempt to push the same resources several times. However, Silk Performer stores the pushed resources only once and notifies the server that additional pushes are not required. If the server has already started pushing resources before it receives the notification, the related traffic will not be measured by Silk Performer.
- If Silk Performer detects a resource that has been pushed by the server but is actually not required during a `WebPage*` function call, a warning displays in the virtual user output and in TrueLog Explorer. Such a warning indicates a server misconfiguration, because the server pushes resources that are not required. This again produces unnecessary network traffic.

In some cases, the Silk Performer push support might differ from the behavior of real browsers. Browsers have different push cache implementations, because there is no official push cache specification.

HTTP/2 in TrueLog Explorer

In TrueLog Explorer, you can check the tabs **In Header** and **Out Header** to find out whether a request uses HTTP/2. If a request does use HTTP/2, the header tabs display so-called pseudo-headers such as `:method`, `:scheme`, `:path`, `:authority`, or `:status`. Note that TrueLog Explorer displays the decompressed version of the headers, because the compressed headers are not human-readable.

When examining the **Statistics** tab, you might notice some more differences:

- HTTP/2 uses only one TCP connection to download all content from one host. Because of that, only the first request to a host should contain values for `DNS`, `Connect` and `SSL Handshake`. The following requests to the same host will reuse the existing connection.
- When you investigate the waterfall diagram for an API node with many embedded resources, you might notice that in an HTTP/2 case the embedded resources can start downloading simultaneously without waiting for an idle TCP connection. While individual requests might take slightly longer, in total the page should be downloaded significantly faster with HTTP/2 compared to previous versions.
- For HTTP/1.1 traffic, TrueLog Explorer displays the number of bytes transferred including SSL and connection overhead for each web node in the TrueLog tree. For HTTP/2 however, concurrent requests to one host share the same TCP connection, thus portions of multiple concurrent requests or responses might be contained in a single SSL frame. To be able to compare HTTP/1.x with HTTP/2 traffic, TrueLog Explorer displays similar statistics, even though the HTTP/1.1 metrics do not suit well to the new protocol version. In detail this means that TrueLog Explorer distributes the SSL overhead across the corresponding requests and responses. Traffic that does not correspond to a specific request/stream, such as HTTP/2 settings frames, is added to the statistics of the subsequent request/response pair. Some measures can seem small. This is caused by the HTTP/2 header compression feature. Especially

when the previous request or response contained many similar headers, this can reduce the number of sent/received bytes for a typical request consisting of mainly headers significantly.
- Due to the nature of HTTP/2, where one frame can contain portions of multiple streams, TrueLog Explorer cannot associate the compressed frame size with particular request/response pairs. Thus, the size of the uncompressed headers and bodies is displayed in the page drill-down view of the **Statistics** tab.

Configuring HTTP/2

HTTP/2 settings can be configured in the **Profile Settings**.

1. In the menu, click **Settings** > **Active Profile** > **Web (Protocol Level)**.
2. HTTP/2 is disabled by default. To enable it, set the **Preferred HTTP version** to `HTTP/2`.

   Note: Make sure you select a **Browser** that supports HTTP/2. To find out which HTTP version a certain browser supports, click **Details** and read the **Supported HTTP version**. Most current browsers do support HTTP/2.
3. **Allow fallback to HTTP/1.1**: You can allow a virtual user to continue with HTTP/1.1 in case the server under test does not support HTTP/2. If the fallback is allowed and actually applied during a test, an informational message will display for each affected server. If the fallback is not allowed, an error message will display.
4. **Allow server push**: You can allow an HTTP/2 server to push resources to the client without an explicit request from the client. If server push is allowed but the server under test does not support the push feature, a warning will display.

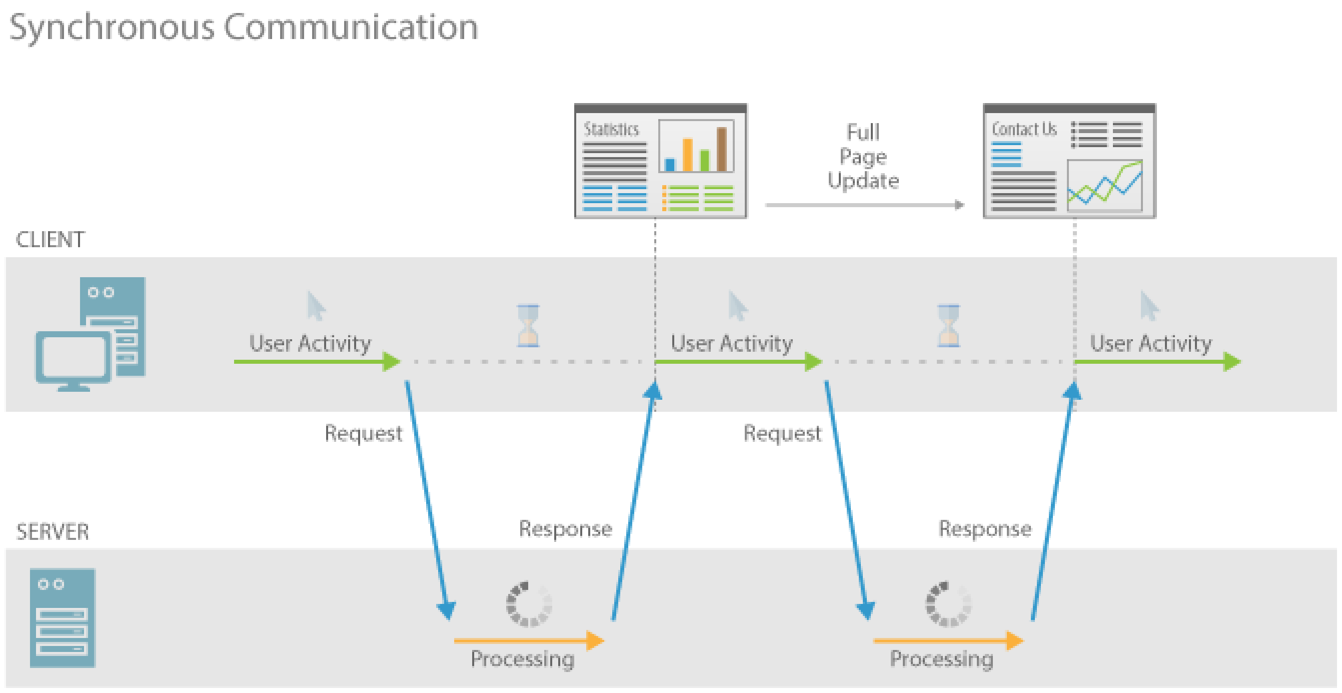
You can also configure these settings in the BDL script using the WebSetOption Function. Note that the settings defined in the BDL script override the options defined in the profile settings.

Testing AJAX Applications

Note: Asynchronous testing functionality for protocol-level record/replay is made for Web applications that use few dedicated asynchronous communication channels. Asynchronous testing on the protocol level is less resource intensive, but it is more challenging to script. The browser-driven testing approach is resource intensive, but it automates the scripting process entirely during recording and is therefore often perceived as an easier way to test AJAX applications.

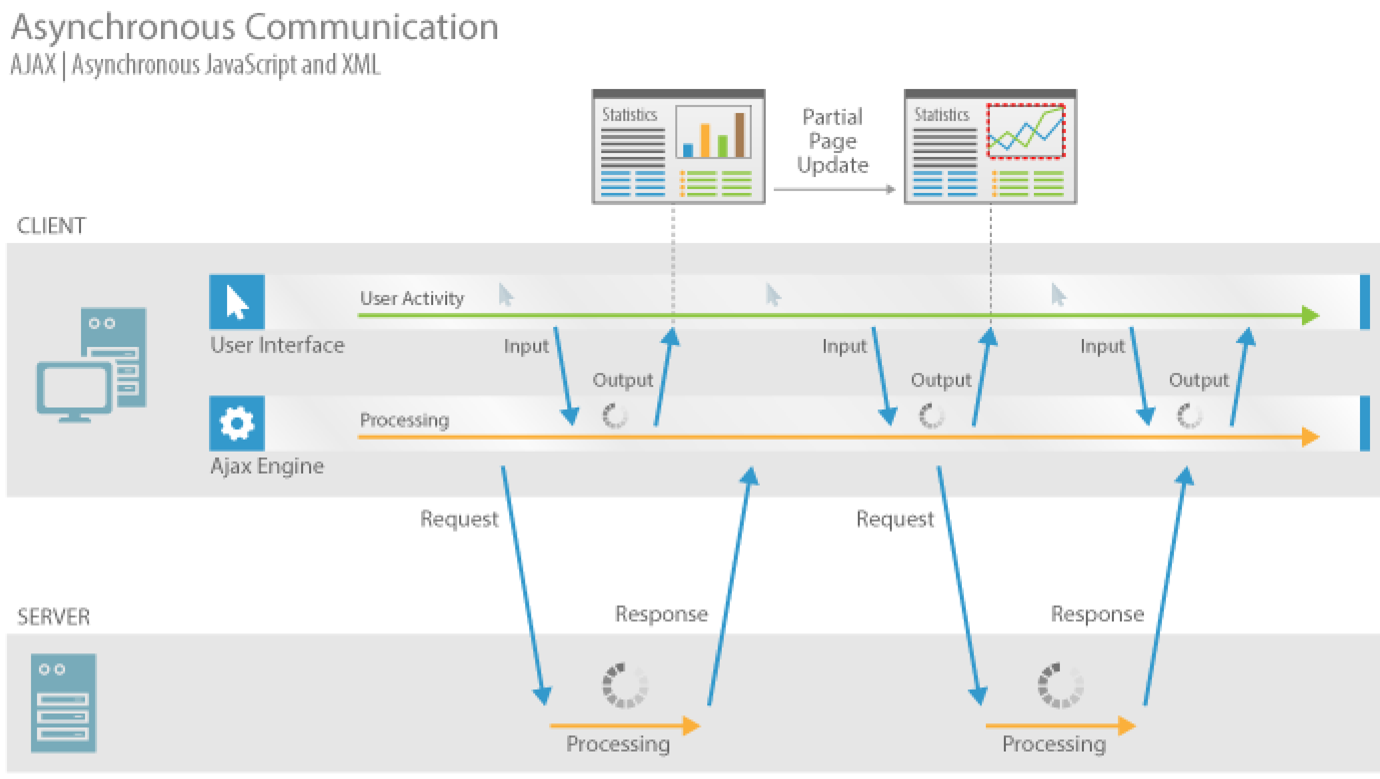
AJAX (Asynchronous JavaScript and XML) is a group of interrelated Web development techniques that are used on the client-side (browser) to create interactive Web applications. With AJAX, Web applications can retrieve data from the server asynchronously in the background without interfering with the display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

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



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

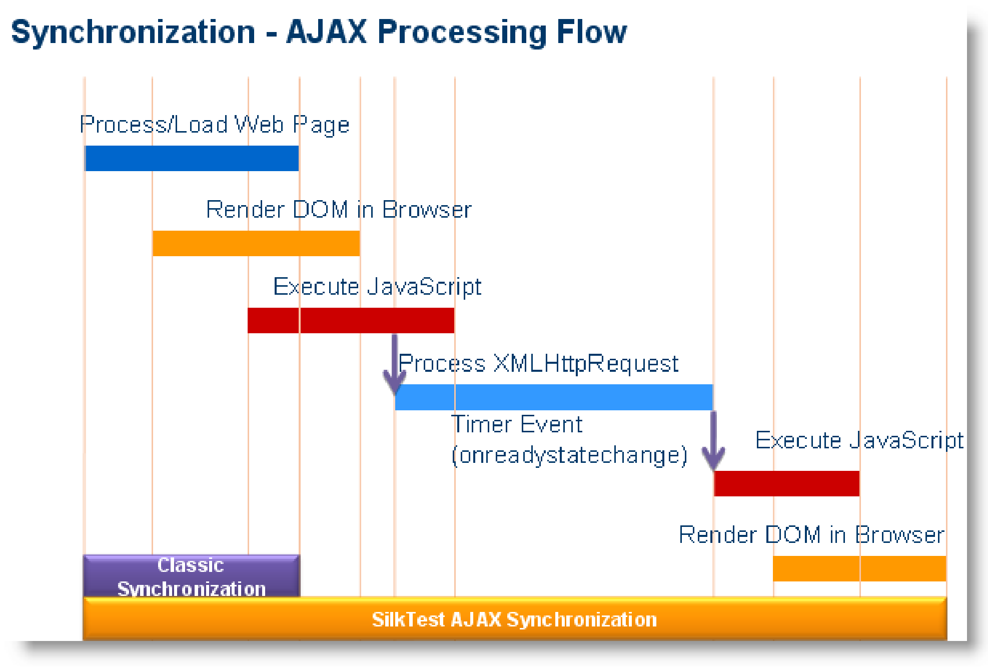
The use of asynchronous requests allows the client Web browser UI to be more interactive and to respond quickly to inputs. In many instances, sections of pages can also be reloaded individually. Users often perceive such applications as being faster and more responsive, even when application state on the server-side has not changed.



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

AJAX Processing Flow

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



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

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

Implications for Automation and Load Testing

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

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

Troubleshooting

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

Identifying DOM Elements in Browser-Driven Testing

Within a browser, all visible and invisible elements of a Web page are represented as a hierarchical tree of objects, the Document Objects Model (DOM). To interact with elements of a Web page, Silk Performer uses the concept of object identifiers, so-called locators. Locators are XPath query strings that identify one or more elements of the current DOM. For additional information about XPath, see *http://www.w3.org/TR/xpath20/*.

All API calls that interact with DOM elements take a locator as an input parameter. For example, the API call `BrowserLinkSelect("//a[@href='www.companyxyz.com']")` uses the locator `//a[@href='www.companyxyz.com']` which identifies the first link that has `www.companyxyz.com` as the value of its href attribute (for example, `<a href="www.companyxyz.com">My Link</a>`).

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

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

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

Unsupported XPath Construct	**Example**
Comparing two attributes to one another	`a[@caption = @href]`
Attribute names on the right side are not supported. Attribute names must be on the left side.	`a['abc' = @caption]`
Combining multiple XPath expressions with `and` or `or`.	`a [@caption = 'abc']` or `.//input`

Unsupported XPath Construct	Example
More than one set of attribute brackets	`div[@class = 'abc'] [@id = '123']` (use `div[@caption = 'abc' and @id = '123']` instead)
More than one set of index brackets	`a[1][2]`
Wildcards in tag names or attribute names	`*/@c?ption='abc'`
Logical operators: `not`, `!=`	`a[@href!='someValue']`, not`[@href='someValue']`
Any construct that does not explicitly specify a class or the class wildcard. For example, including a wildcard as part of a class name.	`//[@caption = 'abc']`

The following list shows wildcard examples that match `//[@caption ='abc def']`:

- `//[@caption ='*c def']`
- `//[@caption ='??c def']`
- `//[@caption ='ab? ?ef']`
- `//[@caption ='ab*ef']`
- `//[@caption ='abc?def']`
- `//[@caption ='abc d??']`
- `//[@caption ='abc d*']`
- `//[@caption ='ab? *']`
- `//[@caption ='?b? d*']`
- `//[@caption ='*c d*']`

Load Testing Web 2.0 Applications (Protocol-Level)

Web Load Testing Overview

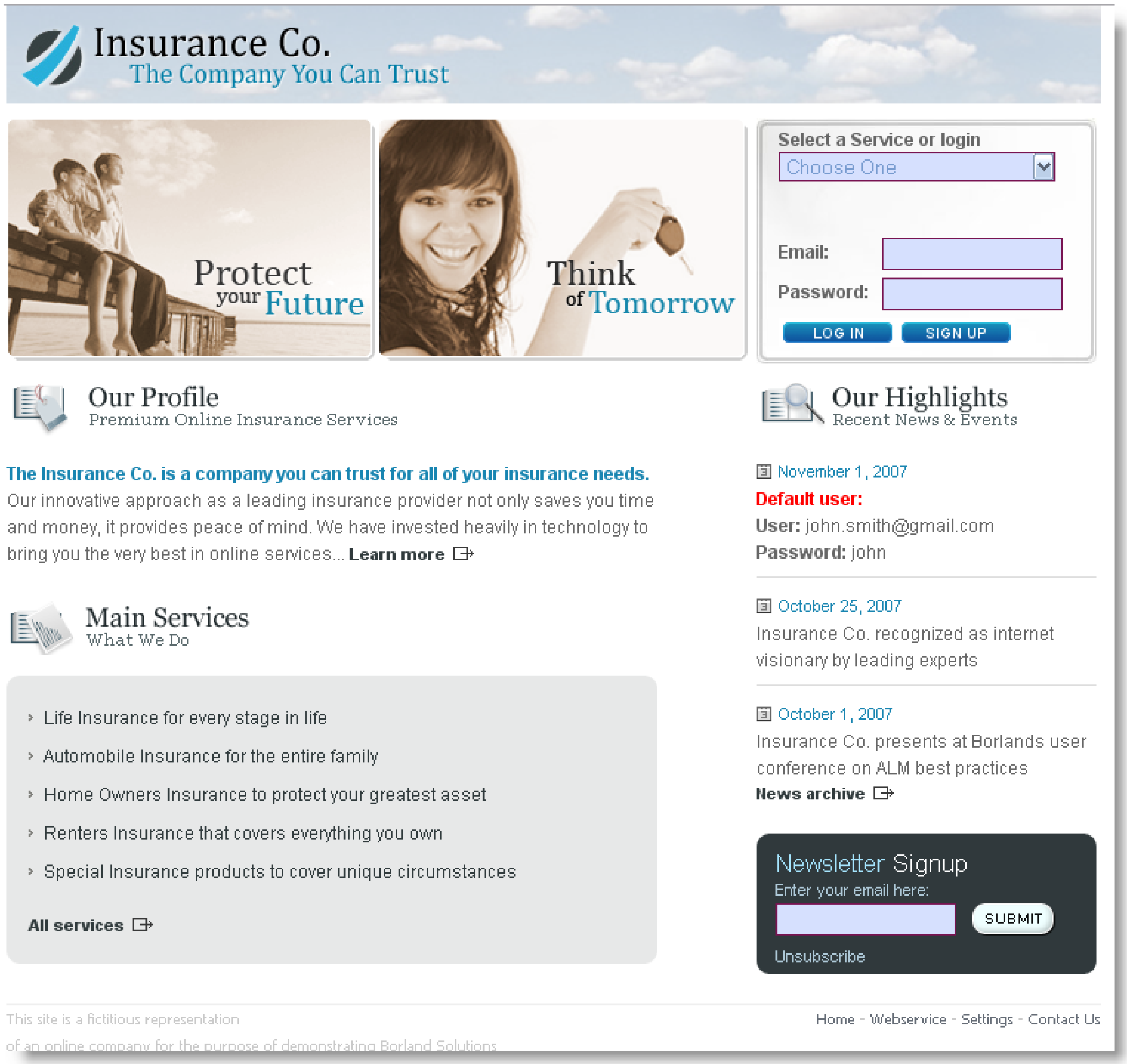
The fastest and easiest approach to test today's modern Web applications is to test them on the protocol level (HTML/HTTP). This approach generates simple scripts that incorporate advanced functionality. When testing a Web application with built-in AJAX logic and testing on the protocol level has proved to be unsuccessful we recommend using browser-driven Web load testing.

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.

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/*.


Insurance Co.
The Company You Can Trust

Protect your Future

Think of Tomorrow

Select a Service or login
Choose One
Email:
Password:
LOG IN
SIGN UP

Our Profile
Premium Online Insurance Services

The Insurance Co. is a company you can trust for all of your insurance needs. Our innovative approach as a leading insurance provider not only saves you time and money, it provides peace of mind. We have invested heavily in technology to bring you the very best in online services... Learn more

Main Services
What We Do

- Life Insurance for every stage in life
- Automobile Insurance for the entire family
- Home Owners Insurance to protect your greatest asset
- Renters Insurance that covers everything you own
- Special Insurance products to cover unique circumstances

All services

Our Highlights
Recent News & Events

November 1, 2007
Default user:
User: john.smith@gmail.com
Password: john

October 25, 2007
Insurance Co. recognized as internet visionary by leading experts

October 1, 2007
Insurance Co. presents at Borlands user conference on ALM best practices
News archive

Newsletter Signup
Enter your email here:
SUBMIT
Unsubscribe

This site is a fictitious representation of an online company for the purpose of demonstrating Borland Solutions

Home - Webservice - Settings - Contact Us

Defining a 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 business transaction (HTML/HTTP)**.
5. Click **Next** to create a project based on your settings.

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

Creating a Test Script

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

Recording a Test Script

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select one of the listed browsers from the **Recording Profile** list, depending on the browser you want to use for recording.
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 dialog opens in minimized form, and the client application starts.
5. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.
6. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.
7. To end recording, click the **Stop Recording** button.
8. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.

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 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 the scripted actions.

Trying Out Your Test Script

1. Click the **Try Script** button on the Silk Performer Workflow bar. The **Workflow – Try Script** dialog appears.
2. Choose a script from the **Script** list box.
3. In the **Profile** list box, the currently active profile is selected (this is the default profile if you have not configured an alternate profile).
   a) To configure simulation settings for the selected profile, click **Settings** to the right of the list box.
   b) To configure project attributes, select the **Project Attributes** link.
4. In the **Usergroup** list of user groups and virtual users, select the user group from which you want to run a virtual user.

   Since this is a Try Script run, only one virtual user will be run.
5. To view the actual data that is downloaded from the Web server during the Try Script in real-time, select the **Animated Run with TrueLog Explorer** check box.

   If you are testing anything other than a Web application, you should disable this option.
6. Click **Run**. The Try Script begins.

All recorded think times are ignored during Try Script runs. The **Monitor** window opens, giving you detailed information about the progress of the Try Script run. If you have selected the **Animated** option, TrueLog Explorer opens. Here you can view the actual data that is downloaded during the Try Script run. If any errors occur during the Try Script run, TrueLog Explorer can help you to find the errors quickly and to customize session information. Once you have finished examining and customizing your script with TrueLog Explorer, your script should run without error.

Analyzing Test Scripts

Silk Performer offers several means of evaluating a test script following the execution of a Try Script run.

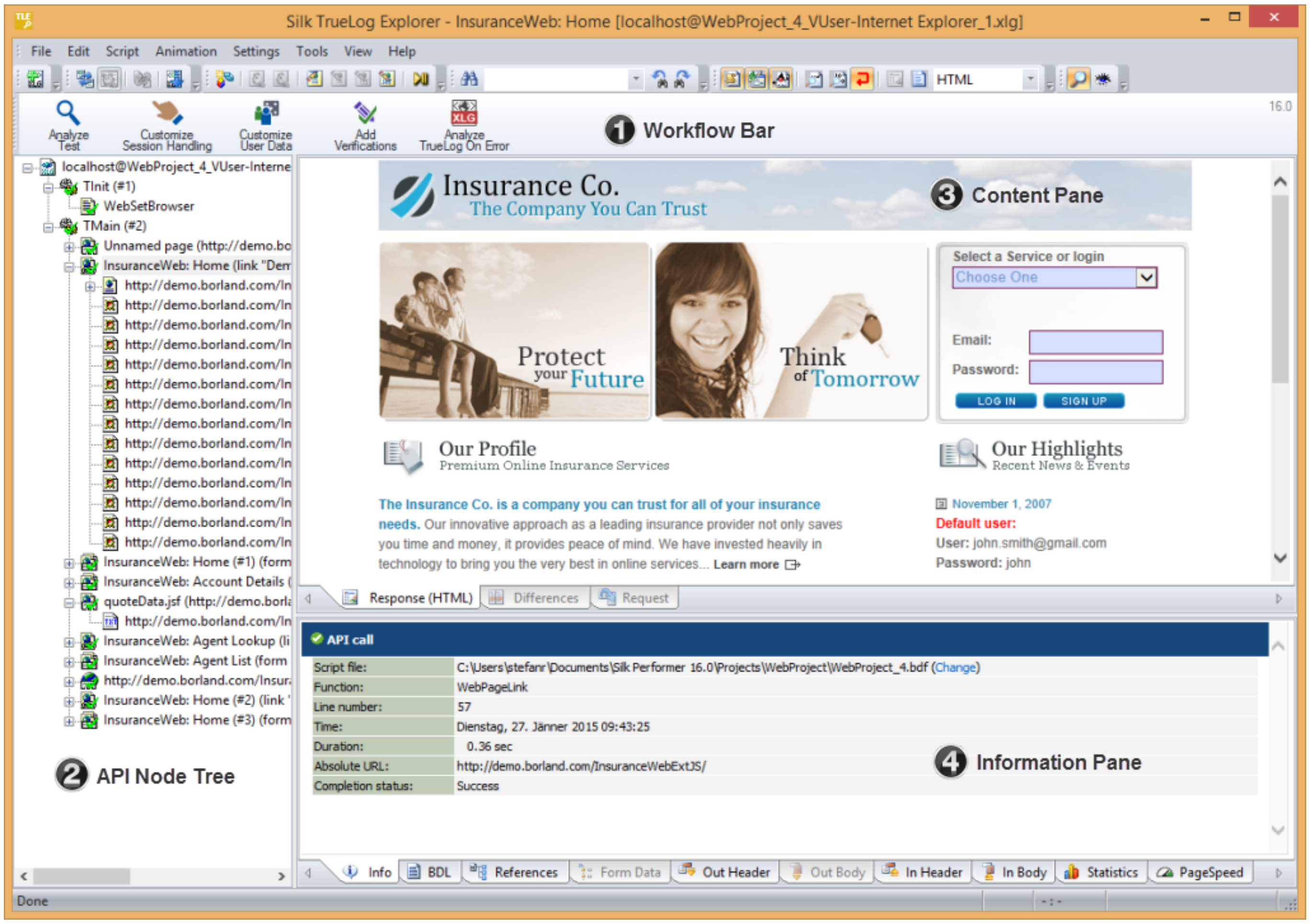
Visual Analysis with TrueLog Explorer

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**.



API call

Script file:	C:\Users\stefanr\Documents\Silk Performer 16.0\Projects\WebProject\WebProject_4.bdf (Change)
Function:	WebPageLink
Line number:	57
Time:	Dienstag, 27. Jänner 2015 09:43:25
Duration:	0.36 sec
Absolute URL:	http://demo.borland.com/InsuranceWebExtJS/
Completion status:	Success

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 Virtual User 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 single virtual user. Data is presented in tabular format.

Note: Because virtual user summary reports require significant processing, they are not generated by default.

To enable the automatic display of virtual user reports at the end of animated Try Script runs, or when clicking the root node of a TrueLog file in the menu tree, check the **Display virtual user report** check box at **Settings** > **Options** > **Workspace** > **Reports** .

Virtual user summary reports include details related to:

- Virtual users
- Uncovered errors
- Response time information tracked for each transaction defined in a test script
- Page timer measurements for each downloaded Web page
- Measurements related to each Web form declared in a test script, including response time measurements and throughput rates for form submissions using POST, GET, and HEAD methods.
- Individual timers and counters used in scripts (Measure functions)
- Information covering IIOP, Web forms, TUXEDO, SAP, and others

Finding Errors in a TrueLog

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

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

1. Open the TrueLog you want to analyze or modify.
2. Click **Analyze Test** on the workflow bar. The **Workflow - Analyze Test** dialog box opens.
3. Click the **Find errors** link. The **Step through TrueLog** dialog box appears with the **Errors** option selected.
4. Click **Find Next** to step through TrueLog result files one error at a time.

   You can select different increments by which to advance through the TrueLog to visually verify that the script worked as intended (**Whole pages**, **HTML documents**, **Form submissions**, or **API calls**).

Viewing Page Statistics

After verifying the accuracy of a test run, TrueLog Explorer can analyze the performance of the application under “no-load” conditions via the **Statistics** tab under the **Information** pane. The **Overview** page details 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. These detailed statistics assist you in pinpointing the root causes of errors and slow page downloads.

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

Time	Description
DNS	The time to translate a host name into an IP address.
Connect	The time to establish a connection to a server.
SSL Handshake	The time to establish a secure layer on an existing connection. The client and server exchange certificates and agree on an encryption technology including related keys to secure data transmission.
Send	The time to hand over the request data from Silk Performer to the operating system. This value does not include the time required to send the request data from the operating system to the server.
Server busy	The time to send the request data from the operating system to the server, the time to calculate the response data on the server, and the time to send the first byte of the response data from the server to the client (including network latency).
Receive	The time to receive the entire response data.
Cache statistics	

Viewing an Overview Page

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

Comparing Record and Replay TrueLogs

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.

Synchronizing Record and Replay TrueLogs

In compare mode you can synchronize corresponding API nodes between replay and record TrueLogs to identify differences between recorded values and replayed values.

Note: This feature is disabled when automatic synchronization of TrueLogs is enabled.

1. Enable compare mode by doing one of the following:
   - Choose **View** > **Compare Mode** .
   - Click the **Compare Mode** button on the toolbar.

2. Open a set of corresponding record and replay TrueLogs.
3. Right-click an API node and choose **Synchronize TrueLogs**. TrueLog Explorer locates the API node in the matching TrueLog that best correlates with the selected API node.

Customizing Test Scripts

Once you have generated a test script with Silk Performer and executed a Try Script run, TrueLog Explorer can help you customize the script in the following ways, among other options:

- **Parameterize input data** – With user-data customization, you can make your test scripts more realistic by replacing static, recorded, user-input data with dynamic, parameterized user data that varies with each transaction. Manual scripting is not required to run such data-driven tests. This feature is available for Web, database, XML, Citrix, SAPGUI, terminal emulation, and Oracle Forms applications.
- **Add verifications to test scripts** – With the **Add Verifications** tool, you can gain insight into data that is downloaded during tests, enabling you to verify that content sent by servers is received by clients. Verifications remain useful after system deployment for ongoing performance management. This feature is available for Web, database, XML, SAPGUI, Citrix, terminal emulation, and Oracle Forms applications. TrueLog Explorer supports bitmap and window verification for applications that are hosted by Citrix servers.

User-Input Data Customization

Without user-input data customization, all simulated transactions are identical and do not account for the variables that are typically experienced in real world environments.

For example, you can customize the user-input data that is entered into forms during testing using the **Parameter Wizard**. The **Parameter Wizard** lets you specify the values that are to be entered into form fields during testing. This enables test scripts to be more realistic by replacing recorded user-input data with randomized, parameterized user data.

Customizing HTML User Data With a New Parameter

Before proceeding, ensure that all static session information has been removed from your test script and that the most recent Try Script run produced a TrueLog that is open in TrueLog Explorer.

With HTML-based applications, the goal of user-data customization is to customize values submitted to form fields.

This task explains the process of creating a parameter based on a random variable.

1. Click **Customize User Data** on the workflow bar. The **Workflow - Customize User Data** dialog box opens.
2. Click the **Customize user input data in HTML forms** link. TrueLog Explorer then performs the following actions:
   - Selects the first **WebPageSubmit API call** node in the menu tree.
   - Opens the **Step through TrueLog** dialog box (with the **Form submissions** option button selected).
   - Displays **Request** view

   Request view shows the page that contains the HTML form that was submitted by the selected `WebPageSubmit` call. When your cursor passes over a form control, a tool tip shows the control's name in addition to its initial and submitted values; an orange line indicates the corresponding BDL form field declaration in **Form Data** view below.
3. Click **Find Next** or **Find Previous** on the **Step through TrueLog** dialog box to browse through all `WebPageSubmit` calls in the TrueLog (these are the calls that are candidates for user-data customization).

   Note: Highlighted HTML controls in **Request** view identify form fields that can be customized.

4. On the **Request** page, right-click the form control that you want to customize and choose **Customize Value**.

   You can replace the recorded values with various types of input data (including predefined values from files and generic random values) and generate code into your test script that substitutes recorded input data with your customizations.

   The **Parameter Wizard** opens.

   With the **Parameter Wizard** you can modify script values in two ways:

   - Use an existing parameter that is defined in the `dclparam` or `dclrand` sections of your script.
   - Create a new parameter based on a new constant value, random variable, or values in a multi-column data file.

   After you create a new parameter, that parameter is added to the existing parameters and is available for further customizations.
5. Click the **Create new parameter** option button and then click **Next** to create a new parameter. The **Create New Parameter** page opens.
6. Click the **Parameter from Random Variable** option button and then click **Next**. The **Random Variable** page opens.
7. From the list box, select the type of random variable that you want to insert into your test script and then click **Next**.

   A brief description of the selected variable type appears in the lower window.

   The **Name the variable and specify its attributes** page opens.
8. Enter a name for the variable in the **Name** text box.
9. Specify whether the values should be called in **Random** or **Sequential** order.

   The **Strings from file** random variable type generates data strings that can either be selected randomly or sequentially from a specified file.
10. In the **File** group box, select a preconfigured data source from the **Name** list box and then click **Next**. The **Choose the kind of usage** page displays.
11. Specify the new random value to use by selecting one of the following choices:

    - **Per usage**
    - **Per transaction**
    - **Per test**
12. Click **Finish**. Your test script now uses the random variable for the given form field in place of the recorded value. The new random variable function appears on the **BDL** page.

Initiate a Try Script run with the random variable function in your test script to confirm that the script runs without error.

AJAX and Script Customization

The Silk Performer Recorder can record and replay Web applications that utilize AJAX (Asynchronous JavaScript and XML) requests. This is possible because Silk Performer recognizes asynchronous AJAX requests and responses that arrive in the form of either XML or JSON within HTML responses. Silk Performer scripts AJAX requests that it encounters as `WebPageUrl` calls.

Silk Performer and TrueLog Explorer support access to values within AJAX requests. This enables script customizations such as input-data parameterization, verification, parsing, and session-information customization within AJAX responses.

Tip: The InsuranceWeb demo application, available at *http://demo.borland.com/InsuranceWebExtJS/*, offers the functionality to switch between JSON and XML serialization methods for testing purposes. From the **Select a Service or login** list, select `Agent Lookup` for JSON or `Agent Lookup (XML)` for XML.

Pretty-Format JSON and XML Data

JSON and XML are data-structure formats commonly used in AJAX applications, REST techniques, and other environments. Silk Performer supports pretty-formatted viewing of XML and JSON-formatted byte streams in BDF scripts. Enhanced rendering of JSON formatted data enables easier customization of string values via TrueLog Explorer's string customization functions.

When JSON-formatted data is recorded or inserted into a BDF script, Silk Performer displays the raw JSON byte stream. Once displayed in JSON format, XML can easily be customized using Silk Performer's Parameter Wizard.

Silk Performer offers the option of viewing JSON data in either pretty-formatted JSON-rendering view or as a raw JSON byte stream.

Understanding JSON

According to JSON.org (www.json.org), "JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language..." "JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language."

Enabling Pretty-Formatted JSON and XML Viewing in TrueLog Explorer

Enable pretty-formatted JSON and XML viewing on the **Response** page if TrueLog Explorer is unable to detect JSON or XML data. If TrueLog Explorer detects JSON and XML data, it is automatically pretty-formatted on the **Response** page.

1. Select a node on the TrueLog menu tree that includes JSON- or XML-formatted data (for example, a `HTTP Post` command node).
2. Click the **Response** tab.
3. Right-click JSON or XML data and choose **Render As** > **JSON** or **Render As** > **XML** to pretty-format the data.

Enabling Pretty-Formatted JSON and XML Viewing in Silk Performer

1. Within a BDL script that includes JSON- or XML-formatted data, right-click within the screen data that you want to view.
2. Select your preferred viewing format from the context menu.
   - Select **Format As** > **JSON** to format the data in enhanced JSON format.
   - Select **Format As** > **XML** to format the data as raw XML.

   You can also select **Format As** > **Auto Format** from the context menu to have Silk Performer determine the best formatting option for screen data types. By default, JSON and XML data is pretty-formatted in BDF scripts.
3. Right-click formatted data strings to access Silk Performer's standard string customization commands.

   Note: Because formatted JSON data is integrated into the Silk Performer code editor, JSON-formatting can be undone/redone using the **Undo**/**Redo** buttons on the toolbar.

   Note: Pretty-formatted JSON and XML data viewing is also available in TrueLog Explorer.

Verifications

Often application errors do not cause erroneous HTTP responses. Instead, the application responds with incorrect data values or with error messages incorporated in the HTML content, such as `A Servlet Exception occurred...` or `Server Too Busy....` A simple check of the HTTP status code will not uncover this class of errors. Therefore, application errors are often overseen if you do not build additional checks into your test script. Verification functions help you to check for application errors that are not simple, standard HTTP errors.

With verification built into your test scripts, your test evolves from being a simple load test to a becoming a combined load and functionality test. You can use these scripts without major performance penalties, even during large load test scenarios. They will, therefore, be able to detect a new class of application errors—errors which only occur under load, but which would be overlooked by a simple load test script without additional verification checks.

Silk Performer offers three means of enhancing your test scripts with verification functionality:

- Automatically let the Recorder generate verification functions during the recording session.
- Directly enhance the script by coding verification functions manually.
- Apply verifications visually via TrueLog Explorer without the need to write a single line of BDL code. TrueLog Explorer will automatically generate verification functions into your script.

  Tip: Refer to TrueLog Explorer Help for detailed information.

Automatically Generating Verifications During Recording

To enable the automatic generation of verifications during Silk Performer script recording:

1. Within Silk Performer, choose **Settings** > **Active Profile** . The **Profile - [<profile name>] - Simulation** dialog appears.
2. Select **Record** > **Web** in the shortcut list, then click the **Verification** tab.
3. In the **Recording** section, check the **Record title verification** and **Record digest verification** check boxes.
4. Click **OK**.

Title Verification

Title verification offers a simple, automatic means of checking to see if applications return correct Web pages. When retrieving HTML documents, the Recorder can automatically generate title verification functions for each page-level Web API call or low-level Web API call.

Note: Title verification does not work when HTML page titles (contents of the `<TITLE HTML>` tags) are not set, or when they share common names.

Title Verification

```
transaction TMain
begin
  ...
  WebVerifyHtmlTitle("ShopIt - Greetings",
WEB_FLAG_IGNORE_WHITE_SPACE |
    WEB_FLAG_EQUAL | WEB_FLAG_CASE_SENSITIVE, 1,
    SEVERITY_ERROR, bVerifyTitleSuccess1);
  WebPageUrl("http://myHost/shopit");
```

Digest Verification

Digest verification is useful for verifying the content of relatively static HTML pages. Digest verification is not useful for HTML pages that are dynamic or contain session information in hyperlinks, embedded objects, or hidden form fields—and therefore change with each call.

The function calls `WebVerifyDataDigest` and `WebVerifyHtmlDigest` check to see if data that Silk Performer receives during replay differs from the data the Recorder captured during corresponding record sessions. With these function calls, Silk Performer generates digests that contain information about the occurrence of all characters and compares those results with digests generated by the Recorder. This function enables you to see if Silk Performer captures the same data during replay that it captured during recording. The Recorder automatically generates a digest verification function for each page-level Web API call and low-level Web API call that retrieves a document with a content type specified for digest verification (the default content type is text/html).

Digest Verification

```
const
  gsVerDigest_ShopIt_Greetings := "\h014200000B..."
                                  "\h016000500A..."
                                  "\h030001000A..."
                                  "\h0900410020...";

transaction TMain
begin
  ...
  WebVerifyDataDigest(gsVerDigest_ShopIt_Greetings, 162);
  WebPageUrl("http://myHost/shopit", "ShopIt - Greetings");
  ...
end TMain;
```

Enabling Verification Checks During Replay

See verification settings options for a list of possible verification checks that you can enable or disable during script replay.

Tip: Profile settings in scripts can be overridden using the `WebSetOption` BDL function.

1. Within Silk Performer, choose **Settings** > **Active Profile** . The **Profile - [<profile name>] - Simulation** dialog appears.
2. Select **Replay** > **Web** in the shortcut list, then click the **Verification** tab.
3. In the **HTML / XML** and **Data** areas, check which verification checks you want to enable during replay.

Inserting Content-Verification Functions

1. Open the TrueLog you want to analyze or modify.
2. Select a TrueLog API node that includes content that you want to have verified (for example, text or an image).
3. Select the content that is to be verified on the **Source** page.

   Note: This step is not required for page-title and page-digest verification functions.

4. Click **Add Verifications** on the workflow bar. The **Workflow - Add Verifications** dialog box opens.
5. Select a pre-enabled verification:
   - Verify the page title
   - Verify the selected text
   - Verify the selected text in an HTML table
   - Verify the digest
6. Complete the following dialog box.

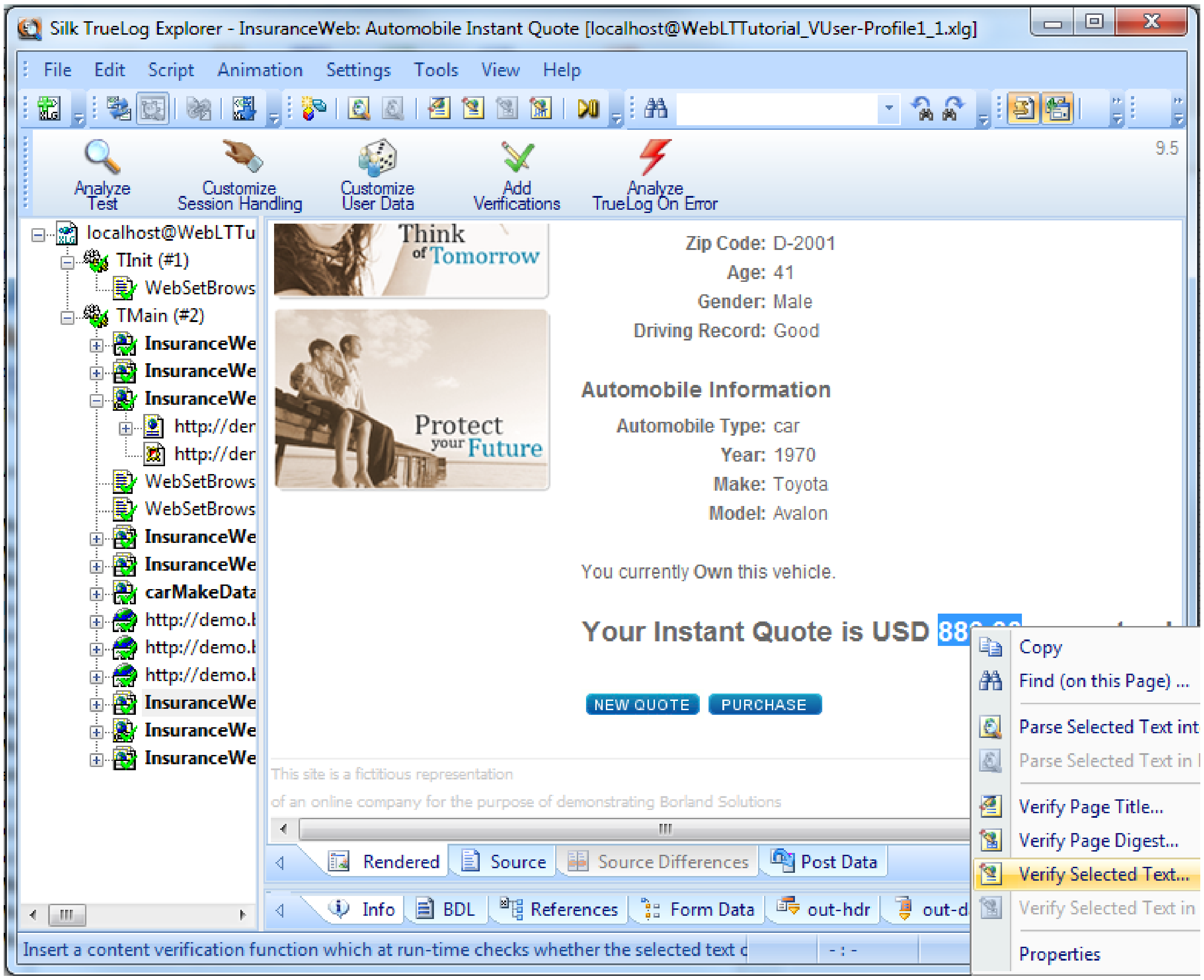
   Specify how verification functions should be inserted into the BDL script.

   Note: Left and right boundaries are automatically identified for you.

7. Repeat the process for each verification you want to add to the BDL script.

8. Click **Yes** on the **Workflow - Add Verifications** dialog box. A Try Script run is initiated.
9. Confirm that verifications have passed successfully.

   API nodes that include verifications are indicated with blue "V" symbols.



Parsing Functions

Similar to the verification functions, parsing functions are used to parse content returned from a Web server and check if the values meet your testing criteria. Different from the verification functions, which basically check the occurrence of a specified input value, the parsing functions parse the content of a server response and return the parsed value in a BDL variable. Typically, you will use parsing functions for the following tasks:

- Parse session IDs and replace static session IDs in the script with parsed, dynamic session IDs to maintain state information in Web applications. This is one of the main fields of application of parsing functions.
- Build enhanced verifications into your script with parsing functions, which cannot be done with verification functions. For example, if you want to verify that the value of column 2 of row 3 in an HTML table is equal to the sum of the values of column 2 of row 1 and column 2 of row 2. By using parsing functions to parse out the three values and compare them in our script, you can build this enhanced verification check.

- Conditional execution of parts of your testing script which depends on the data returned from the server. For example, an HTTP request returns an HTML page which includes: `Results: <nnn> items found`. You want to execute different actions depending on the value of <nnn>. Let's say you want to exit the transaction if `<nnn> = 0`, and you want to go to link **Details** if `<nnn> = 1`, and you want to go to link **Next** if `<nnn> is greater than 0`. To accomplish this, you need to parse the value of <nnn> from the HTML page and script the actions, depending on the value you have parsed out.

Silk Performer offers two means to enhancing your test scripts with parsing functionality:

- Directly enhance your script by coding verification functions manually.
- Apply parsing functions visually in TrueLog Explorer without the need to write a single line of BDL code. TrueLog Explorer will automatically generate parsing functions in your script.

  Tip: Refer to TrueLog Explorer Help for detailed information.

HTML Content Parsing

HTML content parsing functions parse out portions of the rendered, visible HTML content. The HTML content verification and parsing functions allow you to verify and/or parse the textual content that you see in your browser. Apply HTML content parsing functions in the **HTML view** of TrueLog Explorer, which includes the following functions:

- `WebParseHtmlBound`
- `WebParseHtmlTitle`
- `WebParseTable`
- `WebParseResponseTag`
- `WebParseResponseTagContent`

Response Data Parsing

Data parsing functions parse out portions of the response data returned from the server. In cases in which an HTML document is returned from the server, the complete source code of the HTML document is parsed. Apply data verification functions in the **source view** of TrueLog Explorer, which includes the following functions:

- `WebParseDataBound` (this function replaces the deprecated `WebParseResponseData` function)
- `WebParseResponseHeader`

Session Handling

Web applications often use a unique Web session ID so that the Web server is able to handle further client requests.

Refer to TrueLog Explorer Help for detailed information.

Customized Session Handling

Web server applications often generate information at runtime that is necessary in order to identify further client requests. In the response to the browser, the server may include a unique string, commonly known as the Session ID. This string is returned by the browser to the server as a part of each subsequent request, allowing the server to identify the unique Web session of which the request is a part. Generally, Session IDs refer to the method the Web server application uses to identify individual users and to associate this identification with the state of the user session information that the application has previously had with those users.

Session IDs can be sent to the client in a number of ways. Most often you will find them included in cookies, or inside HTML as part of URL's used in hyper links or embedded objects, or in hidden HTML form fields. Session IDs are sent back to the server in cookies, URL's, and HTTP post data.

Session Information Inside Cookies

From the server:

```
Set-Cookie: SessionID=LGIJALLCGEBMIBIMFKOEJIMM; path=/
```

To the server:

```
Cookie: SessionID=LGIJALLCGEBMIBIMFKOEJIMM
```

Session Information in the URL's of HTML Links

From the server:

```
<html>
  ...
  <a href="/ShopIt/acknowledge.asp?
SessionID=LGIJALLCGEBMIBIMFKOEJIMM" >
  Enter Shop
  </a>
 ...
</html>
```

To the server:

```
GET /ShopIt/acknowledge.asp? SessionID =
LGIJALLCGEBMIBIMFKOEJIMM HTTP/1.1
```

Session Information in Hidden Form Fields

From the server:

```
<html>
  ...
  <form action="kindofpayment.asp" method="post" >
  Currently we only accept Credit Cards
  <input type="hidden" name="SessionID"
value="LGIJALLCGEBMIBIMFKOEJIMM">
  <input type="text" name="name" value="Jack " >
  <input type="submit" name="paymentButton" value="Submit">
  </form>
  ...
</html>
```

To the server:

```
POST /ShopIt/kindofpayment.asp HTTP/1.1
...
SessionId=LGIJALLCGEBMIBIMFKOEJIMM&name=Jack&paymentButton=Submi
t
```

When to Use Customized Session Handling

Assuming that a `WebPageUrl` call in your script uses a URL that contains a session ID as part of the query string of the URL. When replaying the script, this hard coded static session ID is sent to the server. Because the session ID does not correctly identify your replayed session—it still identifies the recording session—the replay won’t work correctly. By not replacing static session IDs in your script with dynamic values that had been generated at runtime, your Web application will usually generate errors such as `We are sorry, your session has expired. Please return to the login screen and try again`.

The good news is that with Silk Performer, session customization is not often necessary and, even if manual session customization is needed, Silk Performer's TrueLog Explorer will guide you through the process.

Most often when you record a script, it will work without any modifications necessary for the customization of session handling. Silk Performer therefore uses multiple, highly sophisticated methods that prevent the user from manually handling hard coded session IDs:

- Cookie Management: When your server uses cookies to exchange session information, Silk Performer will automatically handle dynamic session ID values. Because Silk Performer exactly emulates the cookie management of a browser, it will send cookies the same way a browser sends them to your server. No manual interaction is needed for state management.
- page-level web API: Using the Page-level API when recording (which is the default setting) will generate scripts that mainly generate context-full Web API function calls such as `WebPageLink` and `WebPageSubmit`. Context-full Web API calls are working on the HTML level, not the HTTP level, and thus they do not use URL's as parameters. For all context-full API calls, manual session customization is not needed. The page-level API is used if you choose the application type `Web business transaction (HTML/HTTP)` in the **Outline Project** dialog. Therefore it is strongly recommended that you use Silk Performer's page-level API instead of the low-level web API.

Due to the heavy use of client side Java Script for dynamically generating HTML, the Silk Performer Recorder sometimes loses the HTML context and scripts context-less Web API calls. Context-less Web API calls, such as `WebPageUrl` and `WebPageForm`, contain URL's as parameters. In these rare cases, your script may contain hard coded session IDs. You will find them in the URL parameters of Web API calls and in the form fields declared in the `dclform` section of your script.

Context-full script (there is nothing to customize)

```
transaction TMain
  begin
    WebPageUrl("http://lab38/ShopIt/"); // first call is always
context-less
    WebPageLink("Join the experience!");
    WebPageSubmit("Enter", SHOPIT_MAIN_ASP001);
    WebPageLink("Products");
  end TMain;

 dclform
 SHOPIT_MAIN_ASP001:
    "SessionID" := "" <USE_HTML_VAL>, // hidden value:
    "LGIJALLCGEBMIBIMFKOEJIMM"
    // recognized as a hidden form
    // field, the value is taken from
    // the actual HTML form field.
    "name" := "Jack", // changed
    "New-Name-Button" := "" <USE_HTML_VAL> ; //unchanged
    value: "Enter"
```

Script with context-less functions (static session data that needs to be customized is included in the DCLFORM section)

```
transaction TMain
  begin
    WebPageUrl("http://lab38/ShopIt/"); // first call is always
context-less
    WebPageUrl("http://lab38/ShopIt/main.asp", NULL,
SHOPIT_MAIN_ASP001);
    WebPageForm("http://lab38/ShopIt/main.asp",
SHOPIT_MAIN_ASP002);
    WebPageUrl("http://lab38/ShopIt/products.asp");
  end TMain;
```

```
dclform
SHOPIT_MAIN_ASP001:
   "from" := "welcome";

   SHOPIT_MAIN_ASP002:
 "SessionID"       := "LGIJALLCGEBMIBIMFKOEJIMM",
   "name"            := "Jack",
   "New-Name-Button" := "Enter";
```

Customizing Session Handling

The following steps are required to customize session handling.

Refer to TrueLog Explorer Help for detailed information.

1. Identify the session ID that needs to be customized.
2. Search for the first response (Web API call in the script) in which the session ID is sent from the server to the client.
3. Parse out the session ID from the found response of the Web API call into a variable.
4. Replace all occurrences of the hard coded session ID in the script with this variable.

User Profiles

Browser Types

Virtual users in a test can be customized to use any of a wide choice of Web browsers, and of the functionality they embody. The most popular browsers in use today can be emulated. Lesser-known browsers are also available, as are browsers serving mobile device users. Custom browsers can be defined, using different versions of threading technologies and of HTTP.

Bandwidth

Modems are the means that home users typically use to access the Internet. Since a user's modem is sometimes the slowest link in the network communication chain, modems can be simulated in a load test.

Virtual users can be customized to use the bandwidth associated with any of the major connection types in wide use among consumers today. Custom settings can be defined where connections use different bandwidth settings for downstream (from the server to the client) and upstream (from the client to the server) traffic.

Adding a Profile

1. Select **Project** > **New Profile** . The **New Profile** dialog opens.
2. Enter a **Name** for the new profile and click **OK**. The **Profiles** folder expands in the **Project** menu tree and the new profile is available.

Configuring Browser Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, click the **Web** icon.
4. Click the **Browser** tab.

   Use the **Browser type** area to specify browser-specific settings.
5. From the **Browser** list box, select the Web browser you want to use for your simulation.

   The selection you make determines the format of the header information included in your HTTP requests and the threading model used for simulation.

   Note: For mobile Web application testing (iPhone, iPad, Android, Windows Phone or Blackberry) you can change the user agent string used for recording.
6. Click **OK** to save your settings.

Configuring Browser-Simulation Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Web** icon.
4. Click the **Simulation** tab.

   Use the **Simulation** area to set options for realistic simulation of users visiting Web sites.
5. Check the **Simulate user behavior for each transaction** check box to have each virtual users reset their browser emulation after each transaction.

   Depending on the additional option you select, Silk Performer either simulates users who visit a Web site for the first time or users who revisit the site. While users who visit a site for the first time have no persistent cookies stored and no documents cached, users who revisit a page typically have closed their browsers between the Web site visits, have documents cached, and persistent cookies set. Disabling this option lets the virtual user emulate a Web browser that is not closed until the end of the test, thereby reusing cached information throughout multiple transactions.
6. 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.
7. Click the **Revisiting user** option button to generate a realistic simulation of users who revisit a Web site.

   Persistent connections will be closed, and the document history, the non-persistent cookie database, the authentication database, and the SSL context cache will be cleared after each transaction. In such cases, users do not clear the document cache. For more details, review the `WebSetUserBehavior` function in the BDL Function Reference.

   Use the **User tolerance** area to adjust the advanced options of the user tolerance simulation.
8. Click **OK** to save your settings on the **Profile - [<profile name>]** dialog box.

Configuring Monitoring

Before running a test you need to define how Performance Explorer, the Silk Performer server monitoring tool, is to monitor local and remote servers involved in your test. Server monitoring reveals, locates, and assists in resolving server bottlenecks, allowing you to examine the performance of operating systems and application servers.

Three monitoring options are available:

- **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
- **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.
- **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

Defining Monitoring Options

1. Click **Configure Monitoring** on the workflow bar. The **Workflow - Configure Monitoring** dialog box appears.
2. Select one of the following options and click **Next**:
   - **Default monitoring** - This option directs Performance Explorer to monitor a recommended set of data sources based on the application type under test. This is equivalent to enabling the **Automatically start monitoring** and **Use default monitoring template** settings for the Performance Explorer workspace (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).
   - **Custom monitoring** - This option opens Performance Explorer in monitoring mode with the **Data Source Wizard - Select Data Sources** dialog box open, enabling you to manually configure data sources. Your Performance Explorer monitoring project settings will be saved along with your Silk Performer project settings.
   - **No monitoring** - This option enables you to run your test without monitoring of any local or remote servers. With this option the **Automatically start monitoring** setting is disabled (**Settings** > **Active Profile** > **Replay** > **Monitoring** > **Use default monitoring template**).

   (for Default Monitoring and Custom Monitoring only) A confirmation dialog box will notify you if you have logging enabled. Logging may skew your test results.
3. Click **OK** to accept your logging settings or click **Cancel** to adjust your logging options (**Settings** > **Active Profile** > **Results** > **Logging**).
4. *(for Custom Monitoring only)* Performance Explorer starts and the **Data Source Wizard** opens. Complete the steps outlined in the wizard.
5. The **Workflow - Workload Configuration** dialog box appears. Click **OK** to accept your monitoring settings.

Adjusting Workload

Configuring workload is part of the process of conducting a load test. Silk Performer offers different workload models to be used as a basis for your load test. Before configuring workload, you must select the model that best fits your needs.

You can define more than one workload model in your load test project and save them for further usage, but only one workload model can be active at a time. The accepted baseline results are associated with a workload model. If you copy or rename a workload model, the accepted baseline results are copied or renamed accordingly.

Workload Models

Silk Performer provides the following workload models:

- **Increasing** – At the beginning of a load test, Silk Performer does not simulate the total number of users defined. Instead, it simulates only a specified part of them. Step by step, the workload increases until all the users specified in the user list are running.

This workload model is especially useful when you want to find out at which load level your system crashes or does not respond within acceptable response times or error thresholds.

- **Steady State** – In this model, the same number of virtual users is employed throughout the test. Every virtual user executes the transactions defined in the load-testing script. When work is finished, the virtual user starts again with executing the transactions. No delay occurs between transactions, and the test completes when the specified simulation time is reached.

  This workload model is especially useful when you want to find out about the behavior of your tested system at a specific load level.
- **Dynamic** – You can manually change the number of virtual users in the test while it runs. After the maximum number of virtual users is set, the number can be increased or decreased within this limit at any time during the test. No simulation time is specified. You must finish the test manually.

  This workload model is especially useful when you want to experiment with different load levels and to have the control over the load level during a load test.
- **All Day** – This workload model allows you to define the distribution of your load in a flexible manner. You can assign different numbers of virtual users to any interval of the load test, and each user type can use a different load distribution. Therefore, you can design complex workload scenarios, such as workday workloads and weekly workloads. You can also adjust the load level during a load test for intervals that have not started executing.

  This workload model is especially useful when you want to model complex, long lasting workload scenarios in the most realistic way possible.
- **Queuing** – In this model, transactions are scheduled by following a prescribed arrival rate. This rate is a random value based on an average interval that is calculated from the simulation time and the number of transactions per user specified in `dcluser` section of your script. The load test finishes when all of the virtual users have completed their prescribed tasks.

  Note: With this model, tests may take longer than the specified simulation time because of the randomized arrival rates. For example, if you specify a simulation time of 3,000 seconds and want to execute 100 transactions, then you observe an average transaction arrival rate of 30 seconds.

  This workload model is especially useful when you want to simulate workloads that use queuing mechanisms to handle multiple concurrent requests. Typically, application servers like servlet engines or transaction servers, which are receiving their requests from Web servers and not from end users, can be accurately tested by using the queuing model.
- **Verification** – A verification test run is especially useful when combined with the extended verification functionality. This combination can then be used for regression tests of Web-based applications. A verification test performs a specified number of runs for a specific user type.

  This workload is especially useful when you want to automate the verification of Web applications and when you want to start the verification test from the command line interface.

Defining Workload

Prior to the execution of a load test, you must specify the workload model that you want to use and configure its settings. To select a workload model, click **Adjust Workload** in the workflow bar.

1. Click **Adjust Workload** on the workflow bar. The **Workflow - Select and adjust Workload** dialog box appears.
2. Select one of the following workload models by clicking the appropriate option button:
   - Increasing
   - Steady State
   - Dynamic
   - All Day
   - Queuing
   - Verification

3. Click **Next**.
4. Configure specific workload types based on the steps outlined in the Help.

Running Load Tests

In a load test, multiple virtual users are run by means of a test script against a target server to determine the load impact on server performance. A large load test requires an appropriate testing environment on your LAN, including a full complement of agent computers to host the virtual users.

It is essential that you complete the following tasks:

- Set options for the appropriate type of test that is to be run
- Accurately define the required workloads
- Enable the generation of test results to assess server performance

Do not enable complete result-logging during load testing because it might interfere with load-test results. However, the **TrueLog On Error** logging option writes necessary log files to a disk when errors occur, allowing you to inspect replay errors visually.

Running a Load Test

1. Click **Run Test** on the workflow bar. The **Workflow - Workload Configuration** dialog box appears.
2. Click **Run** to start the load test.
3. Click **OK** on the **New Results Files Subdirectory** dialog box.

   (Optional) To specify a name for the results subdirectory, uncheck the **Automatically generate unique subdirectory** check box and enter a name for the new subdirectory in the **Specify subdirectory for results files** text box.

Monitor test progress and server activity by viewing the Silk Performer tabular monitor view and the Performance Explorer graphical monitor view.

Monitoring Load Tests

Detailed real-time information is available to testers while Silk Performer load tests run. Graphic displays and full textual reporting of activity on both the client side and the server side offer intuitive monitoring of the progress of tests as they occur.

Directly from the workbench on which the test is conducted, a tester can view comprehensive overview information about the agent computers and virtual users in the test. A tester can control the level of detail of the displayed information, from a global view of the progress of all the agent computers in the test to an exhaustive detail of the transactions conducted by each virtual user. Progress information for each agent and each user is available in multiple categories. Run-time details for each user include customizable, color-coded readouts on transactions, timers, functions, and errors as they occur.

In addition, real-time monitoring of the performance of the target server is available in graphical form. Charts display the most relevant performance-related information from a comprehensive collection of the Web servers, application servers, and database servers running on all of the OSes most widely used today. Multiple charts can be open at the same time, and these charts can be juxtaposed to provide the most relevant comparisons and contrasts for the tester. A menu tree editor allows for the combination of elements from any data source in the charts. Response times and other performance information from the client application can be placed in the same chart as performance data from the server. This feature enables a direct visual comparison so that one can deterimine the influence of server shortcomings on client behavior.

Monitoring Agent Computers During load Testing

Use the **Monitor** window to view progress while a load test runs. The top part of the window displays information about the progress of agent computers and user groups.

Among the comprehensive number of information options, you can view the following information:

- Status of a particular agent
- Percentage of the test that is complete on an agent
- Number of executed transactions

Monitoring a Specific Agent During load Testing

In the top part of the **Monitor** window, select the specific agent to monitor.

The following information about the virtual users running on the selected agent appears in the bottom of the **Monitor** window:

- Status
- Name of the current transaction
- Percentage of work completed
- Number of executed transactions

Monitoring a Specific Virtual User During load Testing

In the bottom part of the **Monitor** window, right-click the virtual user that you want to monitor and choose **Show Output of Vuser**.

In the **Virtual User** window, Silk Performer displays detailed run-time information about the selected user, such as the transactions and functions the user executes and the data the user sends to and receives from the server.

Tip: Right-click the virtual user area and choose **Select Columns** to select the columns you want to view.

Using a Graph to Monitor Server Performance

1. Click **Confirm Baseline** on the workflow bar. The **Workflow - Confirm Baseline** dialog box opens.
2. Click **Define monitoring options** to specify the settings for receiving online performance data. The **Profile Results** dialog box opens.
3. In the **Profile Results** dialog box, check the **Automatically start monitoring** check box to automatically start monitoring while running a load test and then choose one of the following options:
    - Click the **Use default monitoring template** option button.
    - Click the **Use custom monitoring template** option button to create a customized monitoring template.
4. Click **Create/Edit Custom Monitor Template**. Performance Explorer appears.
5. Close all monitor windows that you are not currently using.
6. Click **Monitor Server** on the Performance Explorer workflow bar.

    Alternatively, you can choose **Results** > **Monitor Server** from the menu bar.

    The **Data Source Wizard / Select Data Sources** dialog box opens.
7. Perform one of the following steps:
    - If you are certain of the data sources that the server provides, click the **Select from predefined Data Sources** option button to then select them from the list of predefined data sources.
    - If you are uncertain of the data sources that the server provides, click the **Have Data Sources detected** option button to let Performance Explorer scan the server for available data sources.
8. Click **Next**.

    In the menu tree, expand the folder that corresponds to the OS on which the server and application are running.
9. Select the server application you want to monitor from the list that appears.
    For example, to monitor the OS, select **System**.

10. Click **Next**. The **Connection Parameters** dialog box opens.
11. In the **Connection parameters** area, specify connection parameters such as the host name or IP address of the appropriate server system, the port number, and other data required to connect to the data source.

    The specified data depends on the OS running on the computer that you are monitoring.
12. Click **Next**. The **Select Displayed Measures** dialog box opens.
13. Expand the menu tree and select the factors you want to monitor.
14. Click **Finish**. A Monitor graph shows the specified elements in a real-time, color-coded display of the server performance. Beneath the graph is a list of included elements, a color-coding key, and performance information about each element.

Exploring Test Results

Silk Performer offers several approaches to displaying, reporting, and analyzing test results. Defined measurements take place during tests and can be displayed in a variety of graphical and tabular forms. Options include the following:

- **Performance Explorer**: This is the primary tool used for viewing test results. A fully comprehensive array of graphic features displays the results in user-defined graphs with as many elements as are required. The results of different tests can be compared. There are extensive features for server monitoring. A comprehensive HTML based overview report that combines user type statistics with time series test result information is also available.
- **TrueLog On Error:** Silk Performer provides full visual verification under load capabilities for various application types. It allows you to combine extensive content verification checks with full error drill-down analysis during load tests.
- **Virtual User Report files:** When enabled, these files contain the simulation results for each user. Details of the measurements for each individual user are presented in tabular form.
- **Virtual User Output files:** When enabled, these files contain the output of write statements used in test scripts.
- **Baseline Reports:** A detailed XML/XSL-based report that provides you with a summary table, transaction response-time details, timers for all accessed HTML pages, Web forms, and errors that occurred. This information is available for all user types involved in baseline tests.
- **Silk Central Reports:** Silk Performer projects can be integrated into Silk Central (Silk Central) test plans and directly executed from Silk Central. This allows for powerful test-result analysis and reporting. For detailed information on Silk Central reporting, refer to *Silk Central Help*.

TrueLog On Error

With Silk Performer TrueLog technology, you can find errors that usually occur to only a subset of users when your application is under a heavy load. For most applications, this is the type of load that will most likely be experienced once the application is deployed in the real world. Typical errors include incorrect text on a Web page, incorrectly computed and displayed values, or application-related messages, such as `Servlet Error` or `Server Too Busy` errors. These are not system-level errors and are displayed on Web pages with `HTTP 200` status codes.

TrueLog Explorer provides a view into Silk Performer verification-under-load capabilities with the following features:

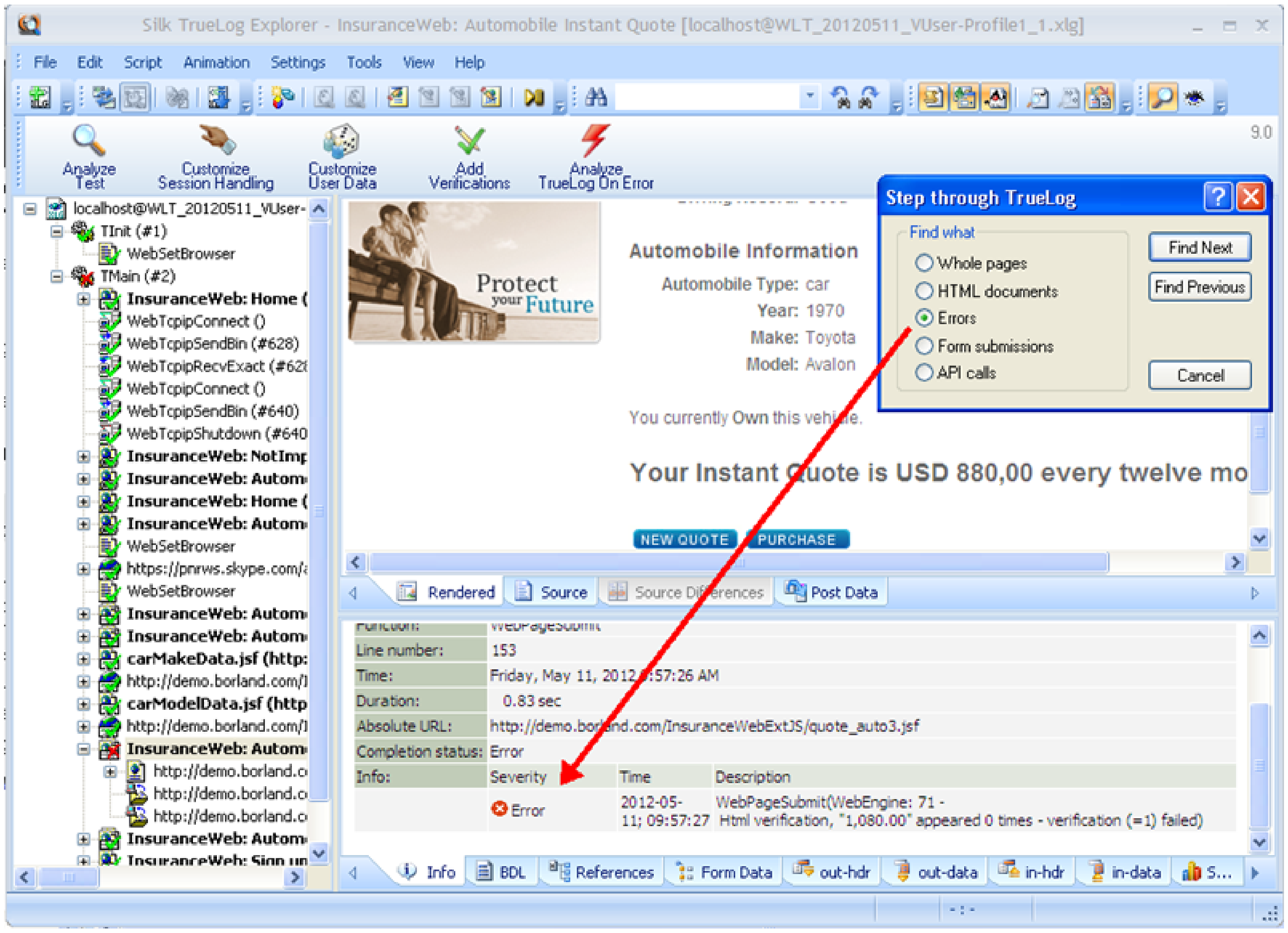
- Visual content verification allows you to visually define the content that is to be verified.
- TrueLog On Error generation and TrueLog On Error analysis allow you to visually analyze errors to identify their root causes.

Viewing Errors in TrueLog Explorer

1. When an error occurs during a load test, you can view the visual content of the TrueLog by clicking the **Explore Results** button on the workflow bar. The **Workflow - Explore Results** dialog opens.

2. Click the Silk TrueLog Explorer link. TrueLog Explorer opens with the **Step Through TrueLog** dialog box active. Select **Errors**.
3. Navigate from one occurrence of an error to the next.

Tip: To display the history of an error, click through the preceding API nodes in the menu tree.



Step through TrueLog

Find what

- Whole pages
- HTML documents
- [✓ Errors]
- Form submissions
- API calls

Find Next | Find Previous | Cancel

```
localhost@WLT_20120511_VUser-
  TInit (#1)
    WebSetBrowser
  TMain (#2)
    InsuranceWeb: Home (
      WebTcpipConnect ()
      WebTcpipSendBin (#628)
      WebTcpipRecvExact (#62
      WebTcpipConnect ()
      WebTcpipSendBin (#640)
      WebTcpipShutdown (#640
    InsuranceWeb: NotImp
    InsuranceWeb: Autom
    InsuranceWeb: Home (
    InsuranceWeb: Autom
    WebSetBrowser
    https://pnrws.skype.com/
    WebSetBrowser
    InsuranceWeb: Autom
    InsuranceWeb: Autom
    carMakeData.jsf (http:
    http://demo.borland.com/I
    carModelData.jsf (http
    http://demo.borland.com/I
    InsuranceWeb: Autom
      http://demo.borland.c
      http://demo.borland.c
      http://demo.borland.c
    InsuranceWeb: Autom
    InsuranceWeb: Sign up
```

Automobile Information

Automobile Type: car
Year: 1970
Make: Toyota
Model: Avalon

You currently Own this vehicle.

Your Instant Quote is USD 880,00 every twelve mo

Line number:	153		
Time:	Friday, May 11, 2012 9:57:26 AM		
Duration:	0.83 sec		
Absolute URL:	http://demo.borland.com/InsuranceWebExtJS/quote_auto3.jsf		
Completion status:	Error		
Info:	Severity	Time	Description
	Error	2012-05-11; 09:57:27	WebPageSubmit(WebEngine: 71 - Html verification, "1,080.00" appeared 0 times - verification (=1) failed)

Performance Explorer Overview

Performance Explorer offers two main capabilities:

- Results Analysis
- Real-Time Monitoring

Results Analysis

With the Results Analysis functionality of Performance Explorer, you can analyze the results of a completed load test by creating charts, tables, and reports. With charts, you can visualize the data that was collected during the load test, and reports as well as tables help you to summarize important data and findings. The basis for all charts, tables, and reports is the data in the .tsd files. During each load test, Silk Performer captures a big amount of data and stores it in several time-series data (.tsd) files. When a load test is completed, you can load the data (the .tsd files) in Performance Explorer and visualize and edit them according to your requirements. All charts are fully customizable, and they can contain as many measures as required. You can open multiple charts using measures from one or multiple tests to show similarities and differences. Performance Explorer provides a variety of templates for charts, tables, and reports. Furthermore, you can place information on client and server performance in a single chart, so that you can directly view the effect of server performance on client behavior. By saving your changes as a template, you can reuse your individual settings.

Real-Time Monitoring

With the Real-Time Monitoring functionality of Performance Explorer, you can monitor a broad range of systems by creating and configuring charts that show the system performance in real-time. It is possible to open multiple charts at the same time, which allows you to watch the performance of two or more systems simultaneously, for example the web server performance and the operating system performance. Adding measures to a chart is easy and intuitive in Performance Explorer: You can drag a single measure or a set of measures from the tree onto the chart.

Overview Report Measurements

The overview report comprises the following sections:

- General information
- Summary tables
- User types
- Custom charts
- Custom tables
- Detailed charts

General information

The general information section includes administrative information in tabular form as well as important load test results in a graphical form.

Administrative information includes the project name, a description of the project, the load test number, a description of the load test, the date of the load test, the duration of the load test, the number of used agent computers, and the number of virtual users that were running.

The charts display the number of active virtual users, response time measurements for transactions, and the number of errors that occur over time. Transaction response times are provided for successfully executed transactions, for failed transactions, and for cancelled transactions.

Additional charts display summary measurements related to the type of load testing project. For example, in the case of Web application testing, response time measurements for Web pages are presented in a graph.

Summary tables

This section contains summary measurements in tabular form, that is, aggregate measurements for all virtual users. The first table provides general information, such as the number of transactions that were executed and the number of errors that occurred. All the following tables provide summary information relevant to the type of application that was tested.

User types

For each user group, this section provides detailed measurements in tabular form. The measurements include transaction response times, individual timers, counters, and response time and throughput measurements related to the type of application that was tested (Web, database, CORBA, or TUXEDO). In addition, errors and warnings for all user groups are listed.

Custom charts

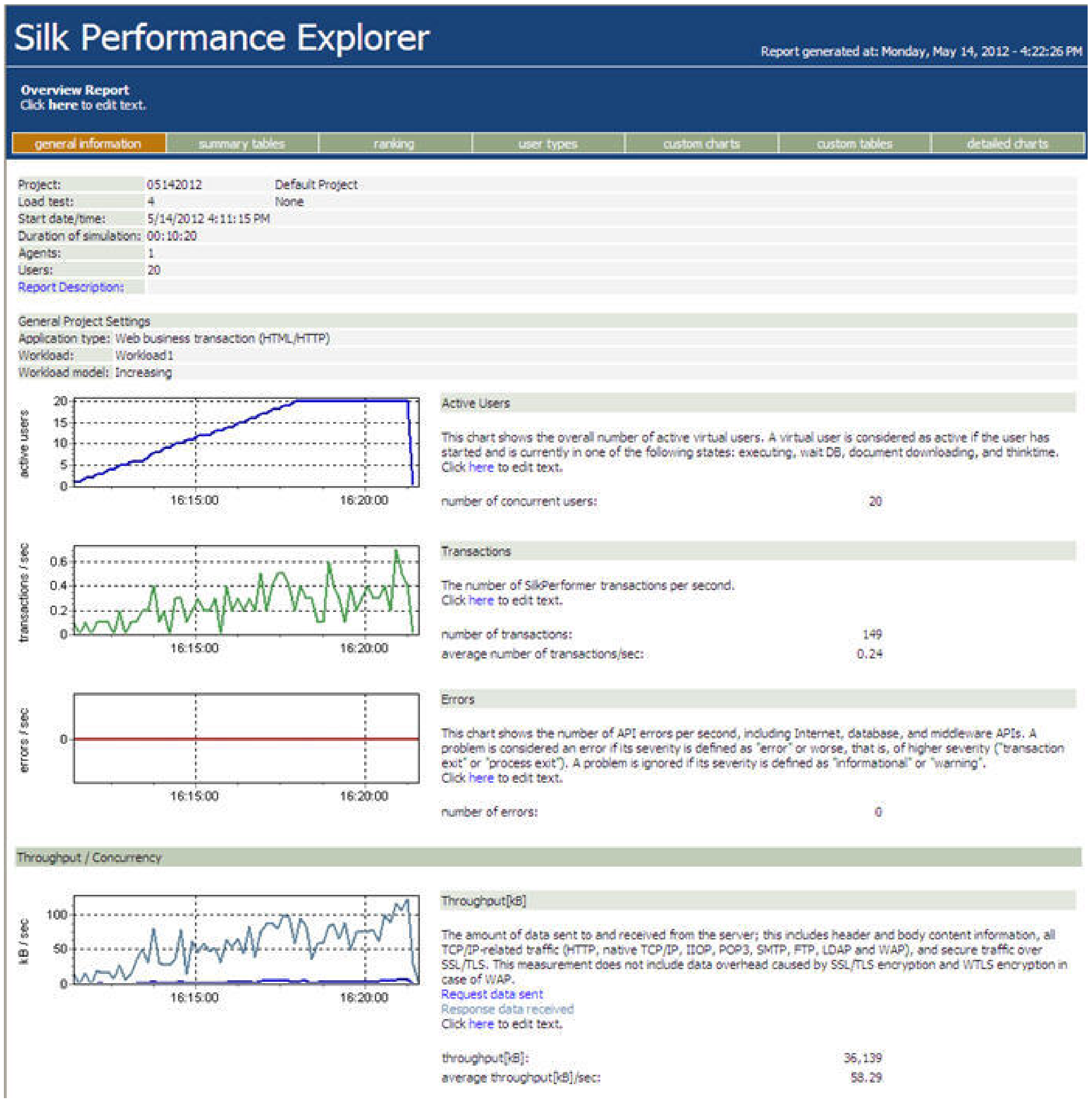
This section contains graphs that you have added manually. You can add charts to and remove charts from this section at any time. You can save your changes as a template to be displayed for every summary report.

Custom tables

This section contains tables that you have added manually. You can add tables to and remove tables from this section at any time. You can save your changes as a template to be displayed for every summary report.

Detailed charts

This section provides enlarged versions of the charts included in the report. Click a reduced version of a chart to jump to the enlarged version, and vice versa.



Report generated at: Monday, May 14, 2012 - 4:22:26 PM

Overview Report
Click here to edit text.

Project:	05142012	Default Project
Load test:	4	None
Start date/time:	5/14/2012 4:11:15 PM	
Duration of simulation:	00:10:20	
Agents:	1	
Users:	20	
Report Description:		

General Project Settings

Application type:	Web business transaction (HTML/HTTP)
Workload:	Workload1
Workload model:	Increasing

Active Users

This chart shows the overall number of active virtual users. A virtual user is considered as active if the user has started and is currently in one of the following states: executing, wait DB, document downloading, and thinktime.
Click here to edit text.

number of concurrent users:	20

Transactions

The number of SilkPerformer transactions per second.
Click here to edit text.

number of transactions:	149
average number of transactions/sec:	0.24

Errors

This chart shows the number of API errors per second, including Internet, database, and middleware APIs. A problem is considered an error if its severity is defined as "error" or worse, that is, of higher severity ("transaction exit" or "process exit"). A problem is ignored if its severity is defined as "informational" or "warning".
Click here to edit text.

number of errors:	0

Throughput / Concurrency

Throughput[kB]

The amount of data sent to and received from the server; this includes header and body content information, all TCP/IP-related traffic (HTTP, native TCP/IP, IIOP, POP3, SMTP, FTP, LDAP and WAP), and secure traffic over SSL/TLS. This measurement does not include data overhead caused by SSL/TLS encryption and WTLS encryption in case of WAP.
Request data sent
Response data received
Click here to edit text.

throughput[kB]:	36,139
average throughput[kB]/sec:	58.29

Viewing Overview Reports

1. Click the **Explore Results** button on the workflow bar. The **Workflow - Explore Results** dialog appears.
2. Click the **Silk Performance Explorer** button or link. If you have selected the **Generate overview report automatically** option in the **Settings/Options/Reporting** dialog, Performance Explorer opens and displays an overview summary report for the most recent load test. Additionally, you can choose to use a previously stored template for the generation of an overview report in this dialog. This setting is a global setting and will be used with Performance Explorer, regardless of the workload project you are using.

3. If the overview report does not appear automatically, click the **Overview Report** button on the workflow bar.
   a) Navigate to the directory of the load test you are working with and select the corresponding `.tsd` file and then click **Open**.
   b) Click **Next**.
   c) Optionally, in the **Template** field, click **[...]** to navigate to the template file that you want to use.
   d) Click **Finish**.
   e) Depending on the load test that you selected, you may be prompted to confirm that you want to load all relevant files into your project.

   The overview report opens.

Silk Performer Web Context Management

This section explains the concepts that facilitate advanced context management (state information management in Silk Performer). It covers the benefits of automatic context management in Web load testing and the techniques used by Web applications to transfer context information between browsers and Web/application servers. Also included is an exploration of the techniques used by the Silk Performer Web API to facilitate automatic context management and an explanation of how to optimally configure the Silk Performer Web recorder to generate scripts that automatically maintain state information.

Value of Context Management

The HTTP protocol, by its very nature, is stateless. HTTP follows a simple request/response paradigm. The client (the browser or other application/applet that uses HTTP for communication with a server) opens a TCP/IP connection to the server, sends a request, and receives a response from the server. Different requests from the same client to the same server, either on the same TCP/IP connection or on another connection, have no relationship to one another.

Web applications on the other hand need to embed single requests within larger entities such as complete HTML pages, user sessions initiated on Web servers, shopping carts, and registered customers.

To do this there must be some means of transferring state information from the server to the client that allows for the information to be returned back to the server within subsequent HTTP requests so that the server can associate the request with a specific user session (for example, a virtual shopping cart used by a registered customer.) and respond appropriately.

The techniques that Web applications use to transfer state information can be used with any Web browser.

A load-testing project that is to deliver useful results must simulate real users accessing Web servers with real browsers. Otherwise results will be meaningless.

Issues that can arise in the absence of state information

Results may be invalid - Poor context management can result in load-testing scripts in which, amongst other issues, each virtual user logs in to the same session, uses the same account, or uses the same shopping cart. In such cases, the impact on the Web/application server is quite different from a comparable load that might be experienced by a live application. Therefore the test results will be meaningless.

Errors go unnoticed - Web servers often do not use HTTP status codes properly when indicating errors. Instead, HTTP status codes that indicate success are returned within HTML documents describing application-level error conditions (for example, `server overloaded` and `session timed out`). If the next instruction in the script does not refer to previous results (for example, by requiring that a link be present), such application level errors may go unnoticed. There is no way for testers to inspect millions of lines of code in thousands of log files to find such errors.

Therefore, context management support contributes directly to improved error detection by providing "auto-verifying" scripts.

Scripts can not be replayed - While replay errors that arise from the auto-verifying functions of advanced context management can be productive, replay errors that arise from improper context management can cause problems, leading to tedious manual customization and debugging of scripts that waste QA resources.

Advanced context management eliminates such annoyances, leading to improved productivity and reduced testing costs.

Silk Performer Page-Level API

Note: To take advantage of Silk Performer advanced context management techniques, you must use the page-level Web API (the default option when you set up a new `Web business transaction` Web testing project in Silk Performer).

The basic idea behind page-level API is to describe user actions at a higher level of abstraction.

Low-level API describes network interactions in terms of single HTTP requests. For single Web pages, browsers typically send numerous HTTP requests to servers. Low-level scripts therefore tend to be rather long, with each single function containing a parameter that specifies a URL from which to download. This makes low-level scripts unsuitable for automatic context management. Low-level scripts can also be difficult to read because they do not describe actions from the user perspective.

In contrast, high-level page-level API describes user actions in terms that are familiar to Web users. For example:

- `WebPageUrl` - Enter a URL in the address bar
- `WebPageLink` - Click a link
- `WebPageSubmit` - Submit a form
- `WebPageBack` - Click the **Back** button

One main feature of page-level API that differs from low-level API is that it incorporates an HTML-parser that parses HTML documents and locates embedded objects, frames, links and forms. Embedded objects and frames are downloaded automatically. Links and forms can be used for context-full function calls.

A single function call in a script can download a complete HTML page, even one that includes complex framesets.

Web Page History

Scripts utilizing page-level API emulate histories by logging previously downloaded pages, just as browsers do to enable their **Back** buttons. The maximum number of pages that are tracked for each virtual user can be limited to manage system resources (the default value is 5).

Note however that this limit does not affect the accuracy of load tests because the history's purpose is to facilitate advanced context management. It has nothing to do with network traffic.

If a page needs to be available for a longer period of time than its life-time in the history allows, it can be locked using the `WebPageStoreContext` function. While the page will still be dropped from the history if necessary, it will not be deleted from memory and it can be referred to by a handle that is returned by the `WebPageStoreContext` function. When the page is no longer needed it can be unlocked (and thereby deleted from memory) using the `WebPageDeleteContext` function.

Context-less Functions

Context-less functions do not refer to previous events or results during script execution. Therefore they work without requiring information from previous events. For context-less function calls, all information that is required to perform functions must be specified in scripts. Scripting and recording of context-less functions requires that session and other dynamic information be included in some test scripts.

A typical feature of context-less functions is that they include parameters that specify URLs for download.

Context-less functions in Silk Performer page-level API include:

- `WebPageUrl`
- `WebPageForm`
- `WebPagePost`, `WebPagePostFile`
- `WebPageCustomRequest`, `WebPageCustomRequestBin`, `WebPageCustomRequestFile`

If any of these functions uses a form from the `dclform` section of your test script, the form will use a context-less approach because there is not a corresponding HTML form from which unchanged (for example, hidden) values can be taken. All form values must be specified in the script.

Context-full Functions

In contrast to context-less functions, context-full functions are designed to use results from previous function calls. How results are to be used is specified in a way that does not incorporate dynamic data.

Context-full functions in Silk Performer page-level API include:

- `WebPageLink`
- `WebPageSubmit`
- `WebPageSubmitBin`

The `WebPageLink` function downloads the target URL of a link.

Example HTML code:

```
<a href=http://www4.company.com/store/5423/account>Edit Account</a>
```

The above link contains load balancing information (`www4`) and a session id (`5423`) in the target URL.

Assume the user clicks this link. This can be modeled in BDL using the `WebPageUrl` function:

`WebPageUrl("http://www4.company.com/store/5423/account");`

The problem with this is that the dynamic components of the URL are hard-coded into the script.

Alternatively, the `WebPageLink` function can be used: `WebPageLink("Edit Account");`

This solution is better because the Silk Performer replay engine will, using its HTML parser, use the actual URL that is associated with the link name **Edit Account** during replay. While it is still possible that a link name can change dynamically, it is far less likely than having a URL change.

The `WebPageSubmit` function submits a form. By doing so, it combines a form definition from a previously downloaded HTML document with a BDL form defined in the script.

```
<form action="/cgi-bin/nav.jsp"
      name="frmNav"
      method="post"
      target="basketframe">
  <input type=input  name="quantity"
                     value="1">
  <input type=hidden name="BV_SessionID"
           value="@@@@1245417051.1003814911@@@@">
  <input type=hidden name="BV_EngineID"
           value="dadccfjhgjehbemgcfkmcfifdnf.0">
</form>
```

Now suppose the user changes the quantity from `1` to `3` and submits the form.

This can be modeled in BDL using the context-less `WebPageForm` function with a corresponding form definition:

```
WebPageForm("http://www4.company.com/cgi-bin/nav.jsp", FORM_BASKET_1);
...
dclform
  FORM_BASKET_1:
    "quantity"    := "3",
    "BV_SessionID" := "@@@@1245417051.1003814911@@@@",
    "BV_EngineID"  := "dadccfjhgjehbemgcfkmcfifdnf.0";
```

The problem with this solution is that everything is hard-coded into the script: the URL, which in this case contains load balancing information, and the form fields, which carry session information.

The better solution is to use the context-full `WebPageSubmit` function:

```
WebPageSubmit("frmNav", FORM_BASKET_2);
...
dclform
  FORM_BASKET_2:
    "quantity"     := "3",
    "BV_SessionID" := "" <USE_HTML_VAL>,
    "BV_EngineID"  := "" <USE_HTML_VAL>;
```

The `WebPageSubmit` function references an HTML form from a previous server response by name. The Silk Performer replay engine, using its HTML parser, automatically uses the actual action URL and HTTP method (`GET` or `POST`) to submit the form. The replay engine then uses the values of the `BV_SessionID` and `BV_EngineID` fields from the actual HTML code, so they do not need to be specified in the script.

Form Definitions in BDL

In Silk Performer, the syntax of form definitions in BDL scripts has been enhanced so that it is possible to specify whether individual fields should use values included in scripts, from HTML code, or be suppressed entirely.

Such specification is achieved using syntactical elements called *form attributes*. Form attributes are also used to specify encoding types for form fields.

For forms that are used context-fully using the `WebPageSubmit` function, the following usage attributes are allowed:

- `USE_SCRIPT_VAL` (default)
- `USE_HTML_VAL`
- `SUPPRESS`

`USE_SCRIPT_VAL` means that the field value from the script is to be used for submission. This is the default attribute and can be omitted.

`USE_HTML_VAL` means that the field value from the HTML form definition is to be used. The value in the script is ignored and set to `""`. Form fields in the script are matched by name with form fields in the HTML form definition.

`SUPPRESS` means that even if a field with a specified name is present in an HTML form definition, neither the field's name or its value will be submitted. The field value in the script is therefore meaningless and is ignored.

How the Recorder Loses Context

Saying that the recorder has "lost context" means that the recorder observes an HTTP request that can not be interpreted as an embedded object, a frame, a link, or a form submission.

This section discusses scenarios that can lead to loss of context in the absence of the advanced context management techniques used by Silk Performer.

META Refresh HTML Tag

HTML allows for the specification of meta information within META tags. A popular use of this technique involves specifying redirections or page reloads within HTML code, rather than relying on HTTP headers to do so (for example, HTTP status code `302 Document moved` for redirections).

Example HTML code:

```
<html>
  <title>Login</title>
  <META http-equiv=refresh
```

```
        content="0; URL=http://www4.company.com/user/6543/login.html">
</html>
```

This example also demonstrates that such non-HTTP redirections can be used (much like standard HTTP redirections) to introduce load balancing and/or session information.

The Silk Performer replay engine does not treat this as a redirection, so the recorder must generate a function call for the resulting HTTP request, thereby losing the context.

Note that this behavior is not a bug, but rather an intentional design feature in Silk Performer replay. To understand why automatically treating this as a redirection would be dangerous (in terms of inaccurate replay), consider the following example:

```
<html>
  <head>
    <title>Login</title>
    <META http-equiv=refresh
      content="1; URL=http://www.company.com/no_js_login.html">
  </head>
  <body onload="location='http://www.company.com/js_login.html'>
  </body>
</html>
```

This sample HTML implicitly checks the browser's JavaScript capabilities and redirects to one of two different login pages, based on the JavaScript capabilities of the browser. If the Silk Performer replay engine automatically downloaded the URL specified in the `meta` tag, it would be demonstrating behavior unlike any Web browser and would therefore be inaccurate.

JavaScript and Web Context Management

JavaScript offers Web developers many options for doing things that can not be achieved through HTML techniques, and therefore can not be simulated using the Silk Performer replay engine by simply parsing HTML. This makes JavaScript a leading cause of context loss.

HTTP requests that result from JavaScript actions often can not be described in terms of `WebPageLink` or `WebPageSubmit` functions, therefore the recorder has to generate context-less functions.

This topic offers some examples, but is by no means a complete listing of all scenarios that may be encountered.

JavaScript Redirection and Reload

This example demonstrates how JavaScript can be used to redirect to another URL, thereby introducing state information (load balancing, session IDs, and more).

```
<html>
  <head>
    <title>Login</title>
  </head>
  <body
    onload="location='http://www4.company.com/usr/6543/login.asp'">
  </body>
</html>
```

Embedded Objects Loaded by JavaScript

This example shows how JavaScript can be used to load embedded objects. The Silk Performer HTML parser however does not view these URLs as embedded objects, so the recorder has to generate a script function, otherwise the images will not be downloaded by the replay engine.

Note also that the URLs of these embedded objects are specified relatively. The browser resolves these relative URLs to absolute URLs using the base URL of the HTML document that contains them. So resulting URLs may well contain dynamic information (for example, `http://www4.company.com/usr/6543/right_arrow.gif`.

```
<html>
  <head>
    <title>Login</title>
  </head>
  <body onload="PreLoadImages('right_arrow.gif', 'left_arrow.gif',
'up_arrow.gif', 'down_arrow.gif')">
  </body>
</html>
```

Dynamic HTML

The `document.write(…)` JavaScript function allows you to dynamically change HTML code on the client side. Ad servers commonly use this technique.

Here's an example:

```
document.write('<script language=javascript
  src="http://ads.com/ad/offer.asp?date=' +
       escape(date) + '"></scr'+'ipt>');
```

Form Submissions with JavaScript

JavaScript can modify HTML forms before submitting them by:

- Modifying form field values (even hidden fields)
- Adding form fields
- Removing form fields
- Renaming form fields (equivalent to removing a form field and adding another field in the same position)
- Changing the action URL

Example HTML code (modified form field names):

```
<form name="tabform"
      action="/cgi-bin/tabgui.asp"
      method="POST"
      target=_self>
  <input type=hidden name="session" value="6543">
  <input type=hidden name="tabevent" value="">
  <input type=hidden name="tabeventparam" value="">
</form>
<a href="JavaScript:selectTab('3')">Stock Watch List</a>

function selectTab(tabIndex)
{
  // change value of field #2
  document.tabform.elements[1].value = "select";
  // change name of field #3, originally "tabeventparam"
  document.tabform.elements[2].name = "TabIndex";
  // change value of field #3, now "TabIndex"
  document.tabform.elements[2].value = tabIndex;
  document.tabform.submit();
}
```

The above example demonstrates how JavaScript can submit a form. The form definition in HTML is just a template that is manipulated by JavaScript before it is submitted. The value of the second field ("tabevent", index 1) is changed to "select", the name of the third field ("tabeventparam", index 2) is changed to "TabIndex", and the value is changed to "3".

The following example demonstrates how JavaScript can be used to change the action URL of a form:

```
function SubmitSearch(linkUrl)
{
  document.searchForm.action = linkUrl;
  document.searchForm.submit();
}
..
<form name="searchForm"
      method="POST"
      target=_self>
  <input type=input name="searchString" value="">
  <input type=hidden name="BV_SessionID"
                     value="@@@@1245417051.1003814911@@@@">
</form>
<a href="JavaScript:SubmitSearch('http://my.server.com/search.asp')">
   Search this site</a>
<a href="JavaScript:SubmitSearch('http://my.mirror.com/search.asp')">
   Search mirror site</a>
```

Advanced Context Management Techniques

The Silk Performer page-level API incorporates advanced techniques that allow the writing of context-full scripts that automatically address context management issues. The Silk Performer recorder handles this work automatically.

The purpose of this section is to explain the features that are offered by the page-level API and show how they are used by the recorder to generate context-full scripts that do not require manual customization for context management.

This section makes reference to Silk Performer advanced context management settings, which can be configured in Silk Performer at **Settings** > **Active Profile** > **Web** > **Recording tab**.

WebPageAddURL

The purpose of the `WebPageAddUrl` function is to specify additional files that should be downloaded during the next page-level API call.

Additional files need to be specified when embedded objects will be downloaded by JavaScript or other techniques that are not covered by the Silk Performer HTML parser.

The URL to download can be specified relative to the base URL of the document that is downloaded by the subsequent page-level API call.

Example:

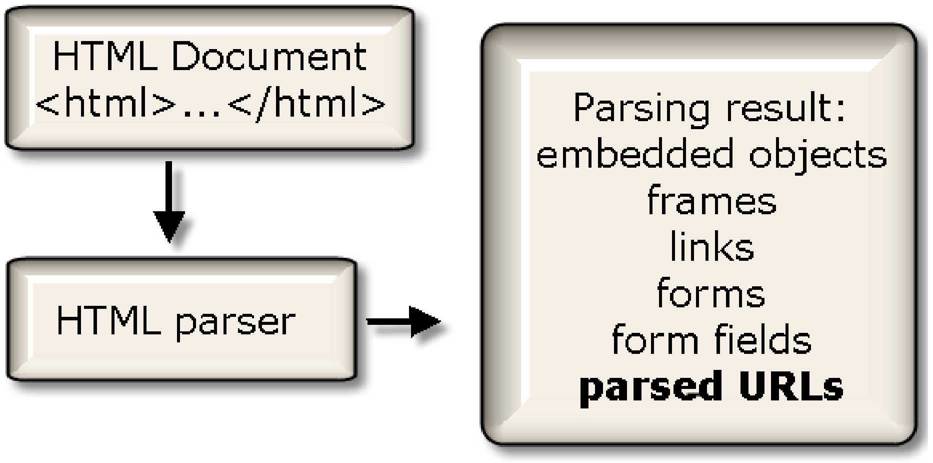
```
WebPageAddUrl("http://www4.company.com/images/mastercard.gif");
WebPageAddUrl("http://www4.company.com/images/visa.gif");
WebPageLink("Choose payment");
The same script in SilkPerformer, assuming the WebPageLink call downloads the
URL http://www4.company.com/user/6543/payment.asp:
WebPageAddUrl("/images/mastercard.gif");
WebPageAddUrl("/images/visa.gif");
WebPageLink("Choose payment");
```

While `WebPageAddUrl` is a context-less function, the use of relative URL's minimizes the risk that a URL parameter will contain state information.

Note: The Silk Performer recorder generates `WebPageAddUrl` calls and relative URL's for `WebPageAddUrl` calls wherever possible.

Parsed URLs

Silk Performer offers an extensible HTML parser that detects parsed URLs (also known as custom URLs). In addition to embedded objects, frames, links and forms, parsed URLs are also a category of HTML element that is detected by the HTML parser.



Extending the HTML parser: WebPageParseURL

To enable the HTML parser to parse custom URLs, you need to specify a parsing rule before the page-level custom URL parsing function. The API function for this purpose is `WebPageParseUrl`. `WebPageParseUrl` works much like the `WebParseDataBound` function.

A parsed URL gets a name (this corresponds to the name of the link). The name is specified in the first parameter of `WebPageParseUrl`. The second and third parameters are the left and right boundaries. The fourth parameter is options (for example, to ignore white spaces when parsing for the boundaries).

A single custom URL parsing specification can result in multiple parsed URLs since parsing does not stop after the first URL is found. More than one `WebPageParseUrl` statement can be placed before a page-level API call, resulting in multiple parsing rules applied concurrently during page download.

Parsing relative URLs

In parsing custom URLs, the HTML parser applies the same rules that are in effect for resolving relative URLs for links and frames. This means that you can parse, for example, only for a filename and receive a complete absolute URL.

Example: The following HTML document was generated by submitting a login form. It has the base URL `http://www4.company.com/user/6543/navigation.asp`:

```
<a href="JavaScript:window.open('account.asp')">Edit Account</a>
```

When the user clicks this link, the browser opens the URL `http://www4.company.com/user/6543/account.asp` in a new browser window.

This can be modeled in Silk Performer as:

```
WebPageParseUrl("Javascript window open", "open('", "'");
WebPageSubmit("login", FORM_LOGIN, "LoginPageTimer");
// now the parsed URL is available under
// the name "Javascript window open" and can be used.
```

Custom URLs, once parsed, can be used in a variety of ways.

Parsed URLs and WebPageLink

A parsed URL that is parsed using the `WEB_FLAG_PARSE_LINK` flag can be used anywhere that a link can be used (for example, for the `WebPageLink` or `WebPageQueryLink` functions). Note that if you do not specify the `WEB_FLAG_PARSE_LINK` and `WEB_FLAG_PARSE_URL` flags, both flags will be in effect by default.

Now the preceding example can be completed:

```
WebPageParseUrl("Javascript window open", "open('", "'");
WebPageSubmit("login", FORM_LOGIN, "LoginPageTimer");
WebPageLink("Javascript window open");
```

Note:

The Silk Performer recorder can generate a `WebPageParseUrl` call and use the parsed URL for a `WebPageLink` call. The recorder does this automatically whenever possible to avoid context-less function calls.

To enable this feature, select the **Dynamic link parsing** checkbox on the **Advanced Context Management Settings** dialog box at **Settings** > **Active Profile** > **Web** > **Recording tab**.

Parsed URLs and context-less functions: WebPageQueryParsedUrl

Parsed URLs can be retrieved and saved in string variables using the `WebPageQueryParsedUrl` function.

Such a string variable can then be used as a parameter for all page-level or low-level functions that require URL parameters, in addition to other purposes (for example, diagnostics output and `StrSearchDelimited`).

The following example HTML document resulted by submitting a login form. It has the base URL `http://www4.company.com/user/6543/navigation.asp`

```
<!--
function ShowContent(url, category, vendor)
{
  top.frames["content"].location.href=
    url + "?cat=" + category + "&vendor=" + vendor;
}
// end of script -->
…
<a href="JavaScript:ShowContent('products.asp', 'HD', 'IBM')">
   hard discs by IBM</a>
<a href="JavaScript:ShowContent('products.asp', 'HD', 'WD')">
   hard discs by Western Digital</a>
<a href="JavaScript:ShowContent('products.asp', 'Mon', 'Sony')">
   Monitors by Sony</a>
```

Assume that the API call that led to this page looks like this:

```
WebPageSubmit("login", FORM_LOGIN, "LoginPageTimer");
```

Now when the user clicks the second link, the browser loads the URL `http://www4.company.com/user/6543/products.asp?cat=HD&vendor=WD` into a frame named `content`. The parsed URLs and WebPageLink approach will not work for modelling this in BDL. That approach would require that parsing boundaries could be found for one of the following strings:

- `http://www4.company.com/user/6543/products.asp?cat=HD&vendor=WD`
- `/user/6543/products.asp?cat=HD&vendor=WD`
- `products.asp?cat=HD&vendor=WD`

But none of these strings occur in the HTML code. Parsing of the URL does not seem possible. Consider the following:

```
WebPageSubmit("login", FORM_LOGIN, "LoginPageTimer");
WebPageUrl(
  "http://www4.company.com/user/6543/products.asp",
  "ProductTimer", FORM_PRODUCT_SELECT);
...
dclform
  FORM_PRODUCT_SELECT:
    "cat"    := "HD",
    "vendor" := "WD";
```

What you get is a context-less function, which, in this example, incorporates dynamic data in the URL. But this can be improved upon. While `http://www4.company.com/user/6543/products.asp?cat=HD&vendor=WD` can not be parsed, the shorter URL `http://www4.company.com/user/6543/products.asp` is now in the script and can be used as a parameter for the `WebPageUrl` function. This URL can be parsed using the boundaries `"ShowContent('"` and `"'"`. This will parse the string `products.asp`, which will, after relative URL resolution, yield the required URL. This parsed URL can be copied into a string variable and the variable can be used rather than the hard coded URL parameter in the script. Thereby the following script is generated:

```
var
  sParsedUrl : string;
..
WebPageParseUrl("ShowContent", "ShowContent('", "'");
WebPageSubmit("login", FORM_LOGIN, "LoginPageTimer");
WebPageQueryParsedUrl(sParsedUrl, sizeof(sParsedUrl),
                  "ShowContent");
WebPageUrl(sParsedUrl, "ProductTimer", FORM_PRODUCT_SELECT);
..
dclform
  FORM_PRODUCT_SELECT:
    "cat"    := "HD",
    "vendor" := "WD";
```

The advantage here is that a context-less function call has been made, in a sense, "semi-context-full." While the query string is still context-less, the URL is parsed and so the dynamic data in the URL can be handled properly.

Note that in other examples it may happen that a URL won't be dynamic, yet the query string will contain dynamic data. In such instances the technique shown here won't deliver improved context management.

Note: The Silk Performer recorder can automatically generate `WebPageParseUrl` calls and query the parsed URLs for use with context-less function calls. The recorder does this whenever this technique can be used in conjunction with a context-less page-level API function call. To enable this feature, select the **Dynamic link parsing** checkbox on the **Advanced Context Management Settings** dialog box at **Settings** > **Active Profile** > **Web** > **Recording tab**.

Changed Action URLs on Form Submission

JavaScript can be used to change the action URL of a form.

```
function SubmitSearch(linkUrl)
{
  document.searchForm.action = linkUrl;
  document.searchForm.submit();
}
...
<form name="searchForm"
      method="POST"
      target=_self>
  <input type=input name="searchString" value="">
  <input type=hidden name="BV_SessionID"
                     value="@@@@1245417051.1003814911@@@@">
</form>
<a href="JavaScript:SubmitSearch('http://my.server.com/search.asp')">
   Search this site</a>
<a href="JavaScript:SubmitSearch('http://my.mirror.com/search.asp')">
   Search mirror site</a>
```

Specifying a different absolute action URL

Assume that this page was downloaded by the function call

```
WebPageLink("Advanced Search");
```

Using the Silk Performer context-full `WebPageSubmit` function is a better choice because it automatically handles the hidden field `BV_SessionID` without having its value in the script.

To facilitate this, Silk Performer has the option of specifying a different action URL to use with the `WebPageSubmit` function.

The function used to do this is `WebPageSetActionUrlAbs`. If this function is called before a call to `WebPageSubmit`, the URL specified in the `WebPageSetActionUrlAbs` function is used rather than the action URL specified in the HTML code.

Using this function, the BDL code above would be rewritten as:

```
WebPageLink("Advanced Search");
WebPageSetActionUrlAbs("http://my.mirror.com/search.asp");
WebPageSubmit("searchForm", FORM_SEARCH);
...
dclform
  FORM_SEARCH:
    "searchString"   := "discount",
    "BV_SessionID"   := "" <USE_HTML_VAL>;
```

Note: The use of `WebPageSetActionUrlAbs` allows you to use the context-full `WebPageSubmit` function rather than the context-less `WebPageForm` function, and so allows you to use the `FORM_SEARCH` form in a context-full way.

The Silk Performer recorder automatically generates the `WebPageSetActionUrlAbs` function when it allows for generation of the function `WebPageSubmit` rather than a context-less function.

To enable this feature, select the **Allow modified action URLs** checkbox on the **Advanced Context Management Settings** dialog box at **Settings** > **Active Profile** > **Web** > **Recording tab**.

Specifying a different parsed action URL

While specifying a different absolute action URL offers an advantage in that it allows for the use of a context-full form, the URL is still context-less and hard coded into the script.

If the action URL also contains state information, the action URL needs to be specified context-fully.

This can be done using the `WebPageSetActionUrl` function. Unlike the `WebPageSetActionUrlAbs` function, the `WebPageSetActionUrl` function receives the name of a parsed URL rather than an absolute URL.

It is possible to parse custom URLs for use with the `WebPageSetActionUrl` function. To demonstrate, the example above can be rewritten as follows:

```
WebPageParseUrl("Form Action", "SubmitSearch('", "'");
WebPageLink("Advanced Search");
WebPageSetActionUrl("Form Action", 2);
WebPageSubmit("searchForm", FORM_SEARCH);
...
dclform
  FORM_SEARCH:
    "searchString"   := "discount",
    "BV_SessionID"   := "" <USE_HTML_VAL>;
```

This example uses the second occurrence of the parsed URL `Form Action` to set the action URL to `http://my.mirror.com/search.asp`. The script can handle state information that may be contained in the action URL.

The Silk Performer recorder can automatically generate calls to the `WebPageSetActionUrl` function and the corresponding call to `WebPageParseUrl`, rather than calling `WebPageSetActionUrlAbs`.

To enable this feature, select the **Dynamic action URL parsing** checkbox on the **Advanced Context Management Settings** dialog box at **Settings** > **Active Profile** > **Web** > **Recording tab**.

Modified Forms

JavaScript can modify forms that are defined in HTML before they are submitted. Removing or renaming individual form fields can be challenging.

Usage attributes for form fields allow for better context management in such situations.

Without form usage attributes, the following example would need to be modeled in a context-less way. When taking advantage of form usage attributes however, it can be modeled context-fully as shown below.

```
<form name="tabform"
      action="/cgi-bin/tabgui.asp"
      method="POST"
      target=_self>
  <input type=hidden name="session" value="6543">
  <input type=hidden name="tabevent" value="">
  <input type=hidden name="tabeventparam" value="">
</form>
<a href="JavaScript:selectTab('3')>Stock Watch List</a>

function selectTab(tabIndex)
{
  // change value of field #2
  document.tabform.elements[1].value = "select";
  // change name of field #3, originally "tabeventparam"
  document.tabform.elements[2].name = "TabIndex";
  // change value of field #3, now "TabIndex"
  document.tabform.elements[2].value = tabIndex;
  document.tabform.submit();
}
```

Assume that the page including this HTML code is a response to the following function call:

```
WebPageLink("My portfolio");
```

Also assume that the page has the base URL `http://www4.company.com/cgi-bin/portfolio.asp`.

The user clicks the link **Stock Watch List**. Without form usage attributes, a corresponding BDL script would look like this:

```
WebPageLink("My portfolio");
WebPageForm("http://www4.company.com/cgi-bin/tabgui.asp", FORM_001);
...
dclform
  FORM_001:
    "session"   := "6543",
    "tabevent"  := "select",
    "TabIndex"  := "3";
```

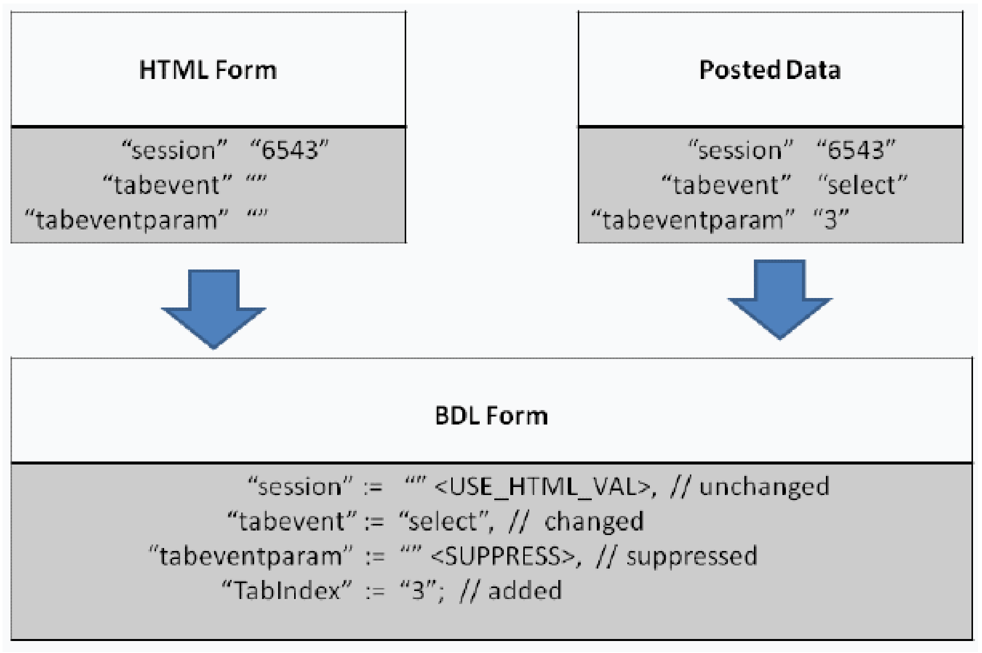
This script uses the context-less `WebPageForm` function. An absolute URL appears in the script, possibly incorporating state information. The form `FORM_001` is context-less, thereby introducing additional state information into the script.

By taking advantage of form field usage attributes, it can be specified that a field with the name `tabeventparam` must not be sent, but that an additional field with the name `TabIndex` should be sent. So the form `tabform` can be submitted in a context-full way using the `WebPageSubmit` function.

Here is the corresponding BDL code:

```
WebPageLink("My portfolio");
WebPageSubmit("tabform", FORM_001);
...
dclform
  FORM_001:
    "session"       := "" <USE_HTML_VAL>, // unchanged
    "tabevent"      := "select",   // changed
    "tabeventparam" := "" <SUPPRESS>, // suppressed
    "TabIndex"      := "3";           // added
```

This version eliminates the state information from the script, thereby providing automatic context management.


HTML Form
"session" "6543"
"tabevent" ""
"tabeventparam" ""
Posted Data
"session" "6543"
"tabevent" "select"
"tabeventparam" "3"
BDL Form
"session" := "" <USE_HTML_VAL>, // unchanged
"tabevent" := "select", // changed
"tabeventparam" := "" <SUPPRESS>, // suppressed
"TabIndex" := "3"; // added

The Silk Performer recorder can automatically generate `WebPageSubmit` calls using HTML forms that do not match submitted form data and then generate appropriate usage attributes for those form fields in scripts. The recorder does this when it allows for a context-full `WebPageSubmit` call to be generated rather than a context-less function call.

To enable this feature, check the **Fuzzy form detection** option on the **Advanced Context Management Settings** dialog box (**Settings** > **Active Profile** > **Web** > **Recording tab** > **View Settings**).

Form Data Encoding

The Silk Performer Web API uses form definitions within the `dclform` section of BDL scripts to specify form data that is to be submitted and query strings to be appended to URLs.

Unsafe characters must be encoded within query strings and HTTP request bodies. To accommodate the various encoding strategies used by different browsers in different situations, Silk Performer allows for the specification of encoding types for form fields.

This is accomplished using the enhanced syntax for form definitions within BDL, which allows for the specification of attributes for forms and form fields.

Encoding Types

The encoding type `ENCODE_FORM` encodes unsafe characters according to the mime type `application/x-www-form-url-encoded`. Blank spaces are encoded with + symbols, unsafe characters are encoded by their hexadecimal values preceded by `%` symbols.

All popular browsers use this encoding type for form submissions, both for URL query strings (HTTP method GET) and HTTP request bodies (HTTP method POST).

The encoding type ENCODE_FORM is the default when no other encoding type is specified in scripts, and therefore does not have to be specified explicitly.

The encoding type ENCODE_ESCAPE encodes unsafe characters according to the escape JavaScript function. This encoding type is similar to ENCODE_FORM. The main difference is that blank spaces are encoded with %20 rather than +.

The encoding type ENCODE_BLANKS encodes blank spaces with %20. No other unsafe characters are encoded. Most popular browsers use this encoding type for the target URLs of links.

The encoding type ENCODE_NONE does not encode any characters. This encoding type is often used when JavaScript issues HTTP requests without using the escape function.

Examples

```
dclform
  AD_SERVER_PARAMS <ENCODE_NONE>:
    "~userpref"     := "12|34|56",
    "~requesttime"  := "01/01/2002";
  FORM_REDIR_LOGIN <ENCODE_NONE> :
    "~language"     := "EN",
    "~logingroup"   := "GUEST",
    "~transaction"  := "TEST",
    "~exitUrl"      := "http://www.myserver.com/goodbye.html";
  LOGIN_FORM_001:
    "username"      := "testuser",
    "password"      := "testpassword",
    "hash"          := "%u2e34!\"" <ENCODE_NONE>,
    "platform"      := "OS: Win2000, Browser: IE5.5" <ENCODE_BLANKS>;
  DOWNLOAD_WHITEPAPER <ENCODE_BLANKS>:
    "filename"      := "Context management.pdf",
    "version"       := "V99";
```

Note: The Silk Performer recorder automatically determines the most suitable encoding type and generates scripts accordingly. The recorder omits the encoding type ENCODE_FORM (the default) to make scripts more readable.

Web Context Management Settings

General Profile Settings

General Silk Performer Web context management profile settings can be configured at **Settings** > **Active Profile** > **Web** > **Recording tab**.

Select **Page-level Web API (HTML/HTTP)** to instruct the recorder to generate a page-level API script.

To have query strings that are part of URL parameters converted into forms for context-less function calls, select **Convert URL query strings to forms**. It is recommended that you select this option as it makes the **Dynamic URL parsing** setting applicable in more cases. If this option is not selected, query strings will not be converted into forms, but rather kept as parts of URLs.

Advanced Settings

To configure advanced Web context management settings, go to **Settings** > **Active Profile** > **Web** > **Recording tab** > **View Settings button**.

Fuzzy form detection - Selecting this option instructs the recorder to generate forms that use the usage attribute SUPPRESS when required. This may allow the recorder to generate the context-full WebPageSubmit function rather than the context-less WebPageForm (for HTTP POST requests) and WebPageUrl (for HTTP GET requests) functions.

Note: It is recommended that you enable **Fuzzy form detection**. If the site under test uses JavaScript to modify forms, this option will offer better context management. If the site does not use JavaScript, this option will not cause any harm.

Allowing modified action URLs - Checking this option instructs the recorder to generate the `WebPageSetActionUrlAbs` function when necessary. This may allow the recorder to generate the context-full `WebPageSubmit` function rather than the context-less `WebPageForm` (for `HTTP POST` requests) and `WebPageUrl` (for `HTTP GET` requests) functions.

Note: It is recommended that you enable **Allowing modified action URLs**. If the site under test uses JavaScript to modify forms, this option will offer better context management. If the site does not use JavaScript, this option will not cause any harm.

Dynamic action URL parsing - Selecting this option instructs the recorder to generate the `WebPageSetActionUrl` function rather than `WebPageSetActionUrlAbs` when it is possible to generate a `WebPageParseUrl` function for the required URL.

Note: It is recommended that you enable **Dynamic action URL parsing**. If the site under test uses JavaScript to modify forms, this option will offer better context management. If the site does not use JavaScript, this option will not cause any harm.

Dynamic link parsing - Checking this option instructs the recorder to generate the `WebPageLink` function for `HTTP GET` requests that do not correspond to a standard HTML link, rather than a context-less function when it is possible to generate the `WebPageParseUrl` function for the required URL.

Note: It is recommended that you enable **Dynamic link parsing**. If the site under test uses JavaScript to modify forms, this option will offer better context management. This option also ensures proper context management when the HTML tag `<meta http-equiv=refresh …>` is used.

Dynamic URL parsing - Selecting this option instructs the recorder to generate the `WebPageParseUrl` and `WebPageQueryParsedUrl` functions when possible to pass parsed URLs as parameters to context-less page-level API functions.

Note: The need for generating context-less API functions is greatly reduced when the preceding four advanced context management options are selected.

Enable **Dynamic URL parsing** if you see context-less API functions in recorded scripts, even if all other advanced context management options are selected and if the context-less functions use URLs that seem to incorporate state information.

Predefined Settings

The **Advanced context management** settings drop-box combo box (available at **Settings** > **Active Profile** > **Web** > **Recording tab** > **Advanced context management**) offers predefined configurations for advanced context management. You can choose a predefined configuration or choose your own custom settings.

The predefined configuration `Level 2` is the default for `Web business transaction (HTML/HTTP)` testing projects.

The other predefined profile settings are detailed in the chart below:

	Disable	**Level 1**	**Level 2 *(default)***	**Level 3**	**Custom**
Fuzzy form detection	-	yes	yes	yes	your choice
Allow modified action URLs	-	yes	yes	yes	your choice
Dynamic action URL parsing	-	-	yes	yes	your choice
Dynamic link parsing	-	-	yes	yes	your choice

	Disable	Level 1	Level 2 *(default)*	Level 3	Custom
Dynamic URL parsing	-	-	-	yes	your choice

Limitations to Web Context Management

You may encounter situations in which advanced context management techniques do not deliver complete, automatic context management. Common reasons include the following:

- A URL can not be parsed by `WebPageParseUrl`. This can happen when a URL is built by JavaScript using string concatenation expressions.
- Link names, form names, or form field names change dynamically.

In such situations, use TrueLog Explorer to customize context management. TrueLog Explorer supports customization of context management using a point and click interface.

Manual Script Editing

Silk Performer test scripts are written in the program's proprietary scripting language, the Benchmark Description Language (BDL), which is a high-level language that resembles Pascal. To edit scripts manually, you must be familiar with BDL and be able to create prototypes of user requests. You must also be able to define the typical components of a test script, including modules, functions, workload definitions, transactions, and Web forms.

Verification and Parsing Functions

All parsing and verification functions must be specified before the Web API calls that will offer the response data that is to be parsed/verified. You can specify multiple parse/verification functions before a Web API call. The order of the parse/verification functions is not relevant (Exception: `WebParseDataBound` and `WebVerifyDataBound` using the flag `WEB_FLAG_SYNCHRON`).

```
WebVerifyHtml("Proper equipment leads to a successful trip",
1, ...);
WebPageLink("ShopIt");
```

Customization of GZIP or ZLIB POST Data

GZIP or ZLIB data in requests

If GZIP or ZLIB data is detected in recorded requests, the `WebSetEncoding` function and the respective parameters are scripted.

GZIP or ZLIB data in responses

If the content encoding header of a response is set to `gzip`, responses are automatically unzipped with the GZIP algorithm. If the content encoding header is set to `deflate`, responses are automatically unzipped with either HTTP 1.1 compatible ZLIB algorithm (RFC1950) or raw deflate (RFC 1951).

Legacy GZIP transformation

To ensure backward compatibility, the GZIP transformation DLL is still supported. For more information on how to enable GZIP transformation, see *Enabling Customization of GZIP POST Data*.

Enabling Customization of GZIP POST Data

Silk Performer's GZIP-transformation functionality enables transparent decompression of GZIP-compressed POST bodies during script recording. Upon replay, POST bodies are recompressed into GZIP format. Without this functionality, GZIP compressed data sent by Web applications via http POST method data would be scripted by the Silk Performer recorder in binary format, making customization of the data nearly impossible. Data customization is however facilitated with Silk Performer's GZIP-transformation functionality.

Transformation is enabled for HTTP requests and responses that have the HTTP header `Content-Encoding` set to `gzip`. If you need to transform data with a different HTTP Content-Encoding header, see the steps below.

Follow the steps below to activate GZIP transformation.

1. Navigate to **Settings** > **Active Profile** > **Record** > **Web** > **Transformation** .
2. From the **Type** drop list, select `GZIP Transformation`.
3. Confirm that the **Transform HTTP requests** option is enabled.
4. Confirm that the **Transform HTTP responses** option is disabled.
5. If you need to transform data with a different HTTP Content-Encoding header, type `AdditionalContentEncodings=<mycustomgzip>` into the **Additional Parameters** field. `<mycustomgzip>` should be the custom Content-Encoding of your application under test.
6. Navigate to **Settings** > **Active Profile** > **Record** > **Web** > **Recording** .
7. In the **Record additional HTTP headers** area, click the **Add** button.
8. Enter `Content-Encoding` into the **Record additional HTTP headers** dialog and click **OK**.

   This is required for accurate compression/decompression of GZIP-encoded request bodies.
9. Back on the **Profile** dialog, click **OK**.

Setting Up Individual IP Addresses

If your application uses a load balancer, you can give the users individual IP addresses by using the *System Configuration Manager*.

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. Click the **IP Address Manager** tab. From the **Adapter** list box, select the network adapter for which you want to configure IP addresses. The list contains all the network adapters to which IP is bound on the computer in question.
3. Use the **IP addresses** area to find out information about the current configuration of the network adapter, and to set up IP addresses for the adapter.
4. To add new IP addresses to the network adapter configuration, click the **Add** button. The **Add IP Addresses** dialog opens.
5. On the **Add IP Addresses** dialog, specify the following:
   a) In the **From IP address** field, enter the IP address you want to add to the network adapter configuration.

      If you specify that you want to add multiple addresses, the System Configuration Manager generates sequential IP addresses, starting with this address.
   b) In the **Subnet mask** field, enter the subnet mask number for the IP addresses.

      Note: This number, combined with the IP addresses, identifies which network the computers are on.

      To add multiple IP addresses, either select the **Number** option and enter the number of addresses you want, or select the **To IP address** option.

      In the latter case, specify the highest IP address you want to add to the network adapter configuration. The System Configuration Manager generates sequential IP addresses, starting with the address you type in the **From IP address** field and ending with this address.
   c) Select the **Do not add conflicting addresses** option to only add IP addresses to the network adapter configuration that do not cause conflicts.

d) When enabling the **Allow CIDR addresses** check box, the verification of the IP address and subnet mask pair is disabled. Classless Inter-Domain Routing (CIDR) has been adopted as a solution to the scaling problem in the Internet. CIDR is not as restrictive as the "normal" IP-Routing.

e) Click **OK**.

Tip: To edit a selected IP address, click the **Edit** button.

6. To save the network adapter configuration to a file, click the **Save** button. By default, the file is stored in the System Configuration Manager home directory.

   To load a network adapter configuration from a file, click the **Load** button and locate the appropriate file.
7. To remove IP addresses, click the **Remove** button.

   Caution: Be cautious about removing multiple IP addresses at once.

   The **Remove IP Addresses** dialog appears.
8. On the **Remove IP Addresses** dialog, specify the following:
   a) In the **From IP address** field, enter the IP address you want to remove from the network adapter configuration.

      If you elect to remove multiple addresses, the System Configuration Manager deletes all successive IP addresses, starting with this address.

   b) In the **Subnet mask** field, enter the subnet mask number for the IP addresses you want to remove.

   c) To remove multiple IP addresses, select either the **Number** or the **To IP address** option. Select the **Number** option to specify the number of IP addresses you want to remove from the network adapter configuration, and enter the number of addresses in the field. The System Configuration Manager deletes that number of successive IP addresses, starting with the address you type in the **From IP address** field.

      Select the **To IP address** option to specify the highest IP address you want to remove from the network adapter configuration. The System Configuration Manager deletes all successive IP addresses, from the address you type in the **From IP address** field to this address.

   d) Click **OK**.
9. If you want to check whether IP routing to a specified computer works for all configured IP addresses, enter the name or IP address of the computer in the **Host name or IP address** field in the **Network** area, then click **Check**. A report from this check will be provided at the bottom of the dialog.
10. Click **OK** to exit the dialog and save your changes.

Web Forms

Web forms are defined in the `dclform` section of test scripts.

Web Form Declaration Syntax

The form declaration is specified by a form name (for example, `FORM_NAME`) which can be any valid identifier string and must be followed by a colon. The left side string specifies the name of the field or parameter. An empty string is allowed for nameless URL parameters, such as image map coordinates. The right value specifies the data value of the field or parameter. These can be global constants, global variables and random variables.

The example below illustrates the required syntax for Web form declarations:

```
dclform
  FORM_NAME:
  "field-name" := R-Value;
```

Web Form String Encoding

All strings used in the forms declaration will be URL encoded when sent to the server. This means that unsafe characters will automatically be converted into a hexadecimal sequence (%xx). Parameters added directly to the URL will not be encoded (for example, `WebUrl("http://host/cgi-bin/doit?MyUnSafeMessage&-?%?-?", 0.0)`. This allows applications to use non-standard URL parameters.

Web Form Script Example

The following example shows a portion of a Silk Performer script for simulating a form submission. Random variables `rsEmail` and `rsPlatform` with two different random functions (`RndFile` and `RndInd`) are defined in the random variables section. With each subsequent call to `WebFormPost` and `WebFormGet`, the random variables are refreshed and contain new random values. These random values are used to generate the URL encoded form string automatically sent within the request to the Web server. `WebFormPost` adds the form content string at the end of the HTTP header while `WebFormGet` adds it at the end of the URL separated by a "?".

```
dclrand
  rsEmail : RndFile
  ("elname.rnd", 20);
  rsPlatform : RndInd("Windows NT" = 0.3;
  "Windows 2000" = 0.6;
  "other" = 0.1);

dcltrans
  transaction TWebFormPost
  begin
    WebFormPost("http://www.comp.com/cgi/FormMail.pl",
CGI_CUST_SUPPORT, nWait);
    WebFormGet("http://www.comp.com/cgi/form",
CGI_CUST_SUPPORT, nWait);
  end TWebFormPost;

dclform
  CGI_CUST_SUPPORT:
  "email" := rsEmail,
  "sitetype" := "business",
  "sitelang1" := "english",
  "platform" := rsPlatform;
```

Cookies

Silk Performer supports record and replay for cookies used in Web applications.

Overview

Cookies enable Web servers to store and retrieve state information on the client computer. A server, when submitting an HTTP object, may also send a "cookie", a state object that the client will store for a specified amount of time. The cookie includes a description that defines a range of URLs for which it is valid. The next time a client reconnects from within this range of URLs, the contents of the cookie will be submitted to the server in addition to the request. Cookies may either be persistent or non-persistent. Persistent cookies are stored in a cookie database on the client computer and are deleted only after a specified expiration date occurs. Non-persistent cookies are maintained only during a single Web browser session and are deleted when the client Web application is closed. To create as realistic a simulation as possible, only persistent cookies are recorded (and thus replayed) by the Internet Recorder.

Cookie Database Functions

For information about cookie database functions, see Cookie functions. Before you use these functions, you should be familiar with cookies as outlined at *http://www.w3.org/Protocols/rfc2109/rfc2109* .

Important Notes

Keep in mind the following when you work with cookies with Silk Performer:

- The Silk Performer cookie database is non-persistent and therefore only exists while a test is running.
- Each virtual user has a cookie database of its own and therefore the contents of the cookie database varies from virtual user to virtual user.
- Although cookies are generally created at server side, you can simulate already existing cookies using the `WebCookieSet` function.

Load Testing Web 2.0 Applications (Browser-Driven)

In addition to facilitating testing of today's modern Web applications on the protocol level (HTTP), Silk Performer enables you to use real Web browsers (Internet Explorer, Chrome, and Firefox) to generate load.

Browser-driven load testing is available in two flavors:

- Re-using a functional test created with Silk Test. This approach applies best if you already have Silk Test keyword test assets that automate functional tests with one of the supported browsers. When importing such assets, Silk Performer wraps them into a BDL stub script to make it executable as part of a load test. For more information, see *Single Session GUI-Level Testing* in *GUI-Level Testing Support*.
- Creating browser-driven load testing scripts from scratch within Silk Performer. With this approach, the BDL scripting language is used to model the user actions. With this option, no Silk Test installation is required. The user actions are recorded within a purpose-built tool named Browser Application. This application hosts an Internet Explorer component and facilitates recording, including adding verifications, defining Time-to-interact elements and visual debugging.

  Note: Recording within Chrome or Firefox is currently not supported. But once recorded within the Browser Application, you can replay your script in Chrome or Firefox.

Note: In terms of licensing, there are no differences. For both types of browser-driven load testing, the license type *SilkPerformerWebVU* is required.

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)`.

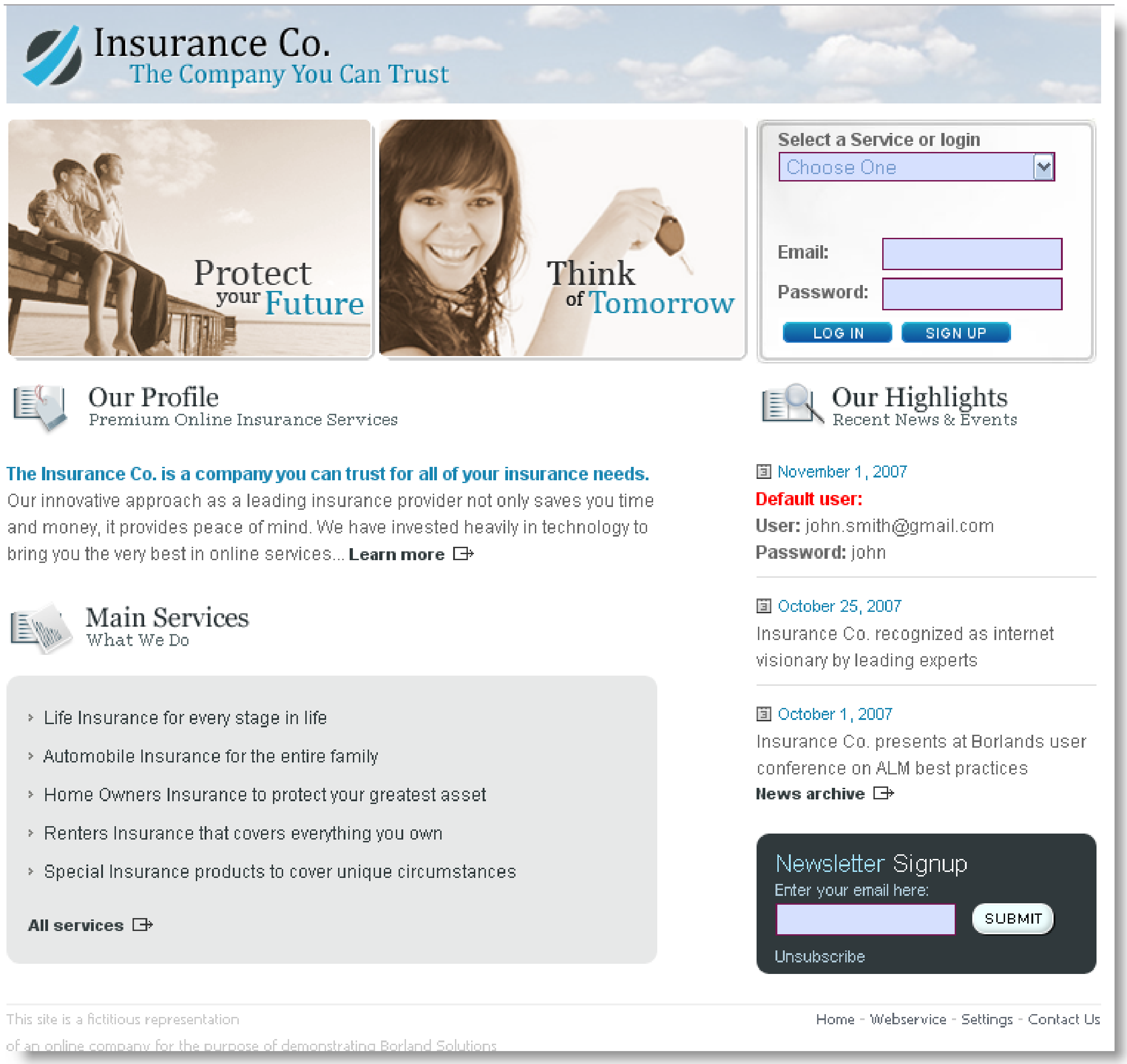
A new tab is created in the Browser Application each time a pop-up window is generated during application recording. Each pop-up window that is encountered results in a tab being created in the Browser Application. Each time you click a tab in the Browser Application during recording a `BrowserActivateWindow`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



The Insurance Co. is a company you can trust for all of your insurance needs.
Our innovative approach as a leading insurance provider not only saves you time and money, it provides peace of mind. We have invested heavily in technology to bring you the very best in online services... **Learn more**

- Life Insurance for every stage in life
- Automobile Insurance for the entire family
- Home Owners Insurance to protect your greatest asset
- Renters Insurance that covers everything you own
- Special Insurance products to cover unique circumstances

All services

November 1, 2007
Default user:
User: john.smith@gmail.com
Password: john

October 25, 2007
Insurance Co. recognized as internet visionary by leading experts

October 1, 2007
Insurance Co. presents at Borlands user conference on ALM best practices
News archive

This site is a fictitious representation
of an online company for the purpose of demonstrating Borland Solutions

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**	Avoids unexpected dialog boxes. **Note:** Depending on your browser version, not all settings may be available.

Tab Name	Option	Configuration	Comments
		• Disable all **Warn...** settings	

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.

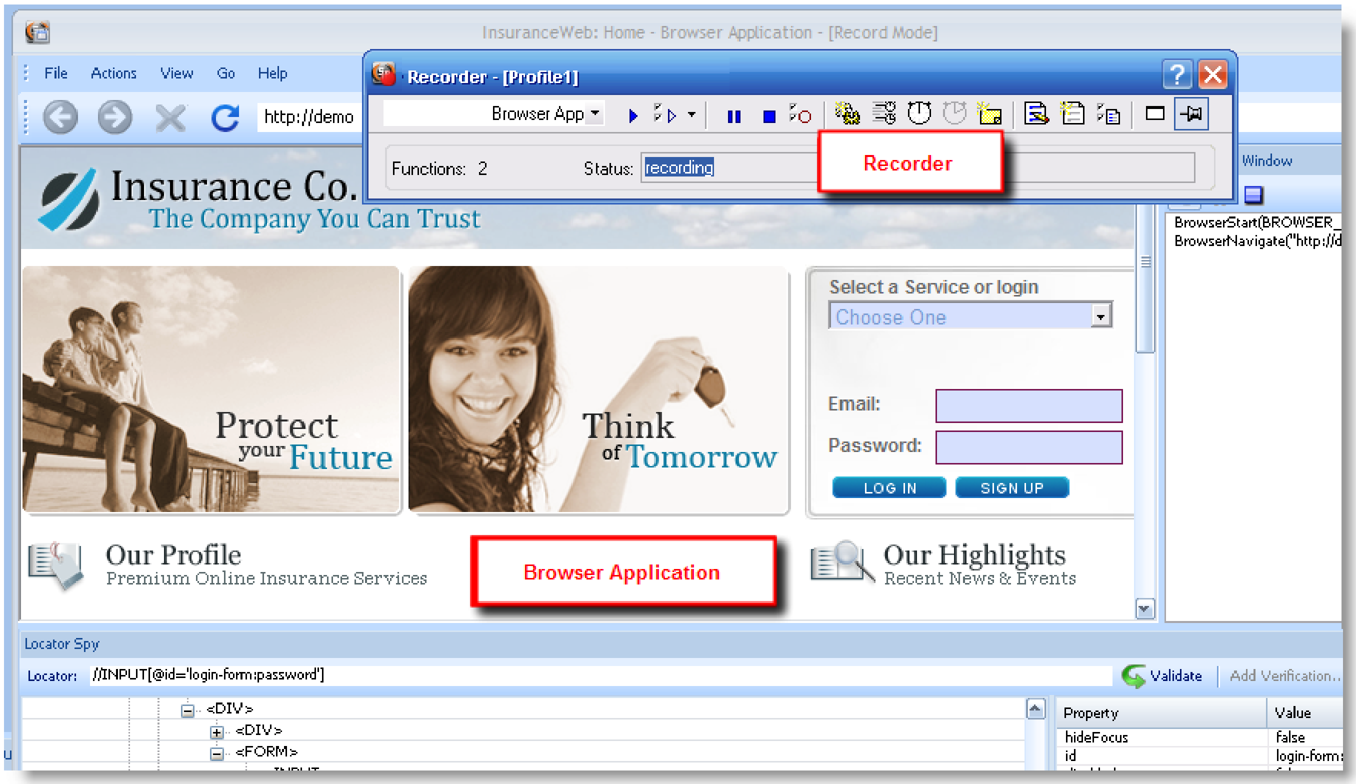
1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select Silk Performer **Browser Application** from the **Recording Profile** list.
3. In the **URL** field, enter the URL that is to be recorded.

   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.



Locator: //INPUT[@id='login-form:password']

```
BrowserStart(BROWSER_
BrowserNavigate("http://d
```

Property	Value
hideFocus	false
id	login-form:

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.

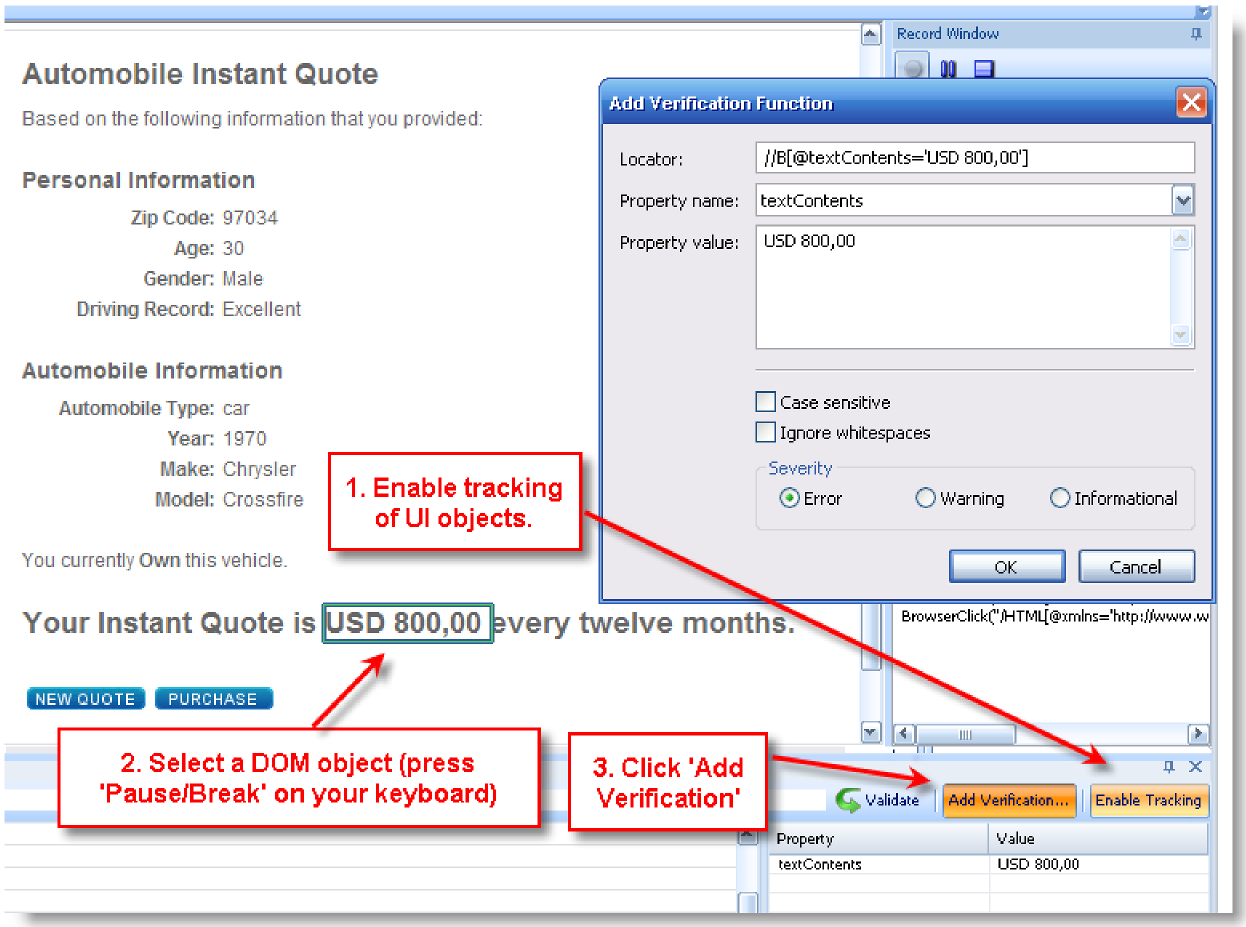
   The locator of the selected UI object appears in the **Locator** text box and the DOM hierarchy is displayed in the tree menu.
2. Click **Add Verification**.

   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.



Automobile Instant Quote

Based on the following information that you provided:

Personal Information

Zip Code:	97034
Age:	30
Gender:	Male
Driving Record:	Excellent

Automobile Information

Automobile Type:	car
Year:	1970
Make:	Chrysler
Model:	Crossfire

You currently Own this vehicle.

Your Instant Quote is USD 800,00 every twelve months.

Add Verification Function

Locator:	//B[@textContents='USD 800,00']
Property name:	textContents
Property value:	USD 800,00

[ ] Case sensitive
[ ] Ignore whitespaces

Severity: [✓ Error] Warning Informational

BrowserClick("/HTML[@xmlns='http://www.w

Property	Value
textContents	USD 800,00

1. Enable tracking of UI objects.

2. Select a DOM object (press 'Pause/Break' on your keyboard)

3. Click 'Add Verification'

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

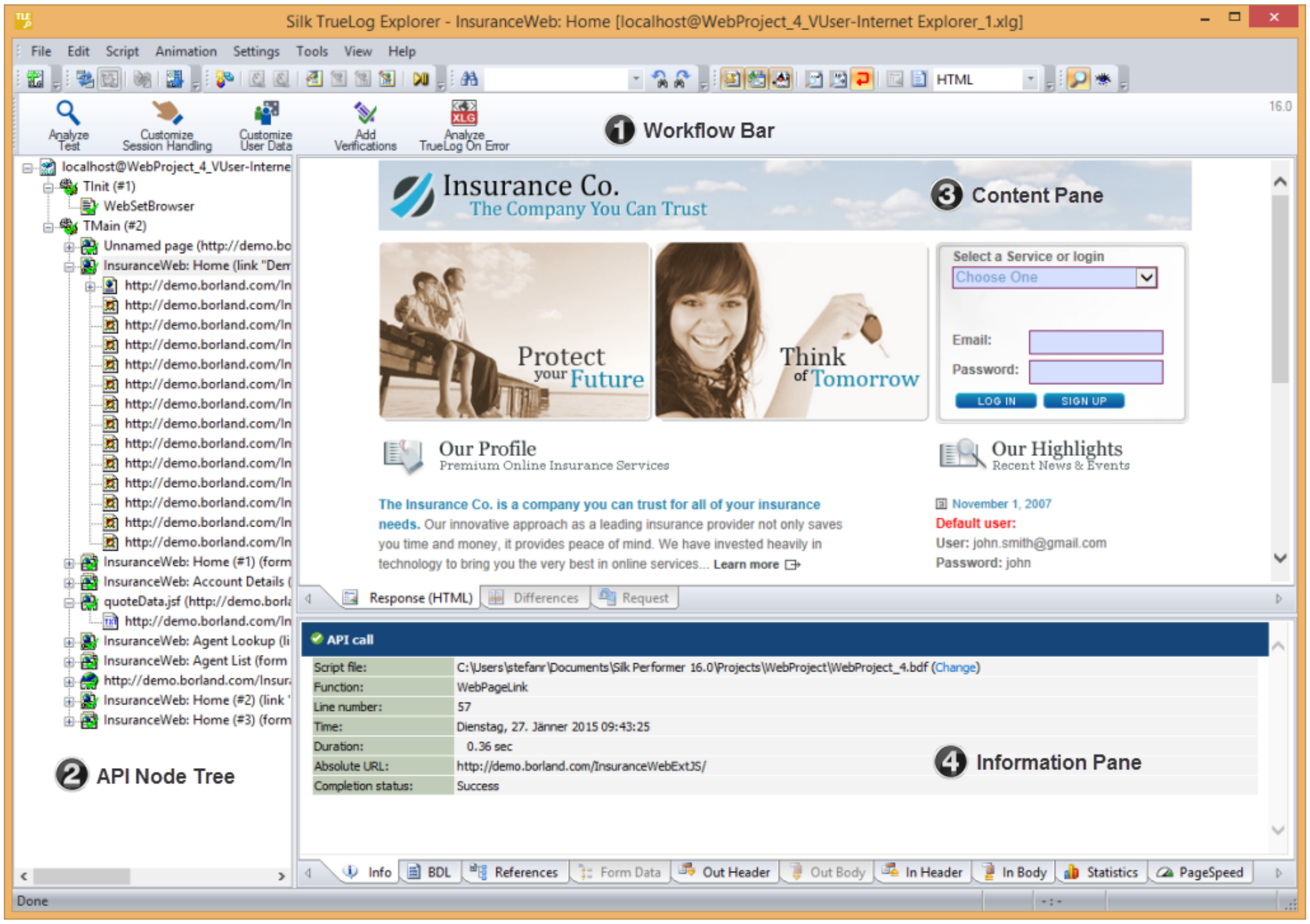
Visual Analysis with TrueLog Explorer

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**.



API call

Script file:	C:\Users\stefanr\Documents\Silk Performer 16.0\Projects\WebProject\WebProject_4.bdf (Change)
Function:	WebPageLink
Line number:	57
Time:	Dienstag, 27. Jänner 2015 09:43:25
Duration:	0.36 sec
Absolute URL:	http://demo.borland.com/InsuranceWebExtJS/
Completion status:	Success

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

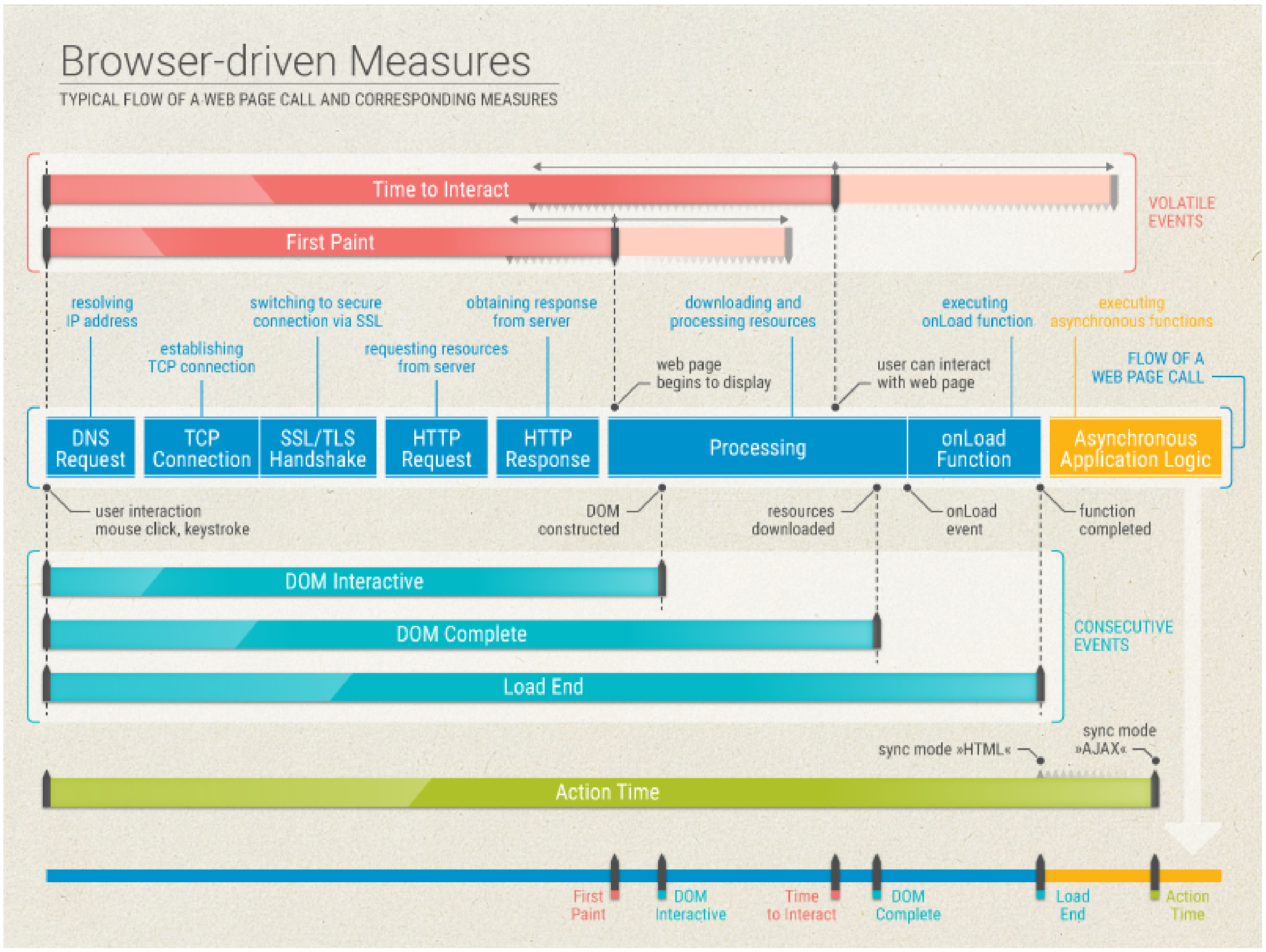
Testing AJAX websites is challenging

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.
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

Measure	Description
	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.

Recommended Internet Explorer settings on agents

When executing browser-driven load tests using Internet Explorer, make sure that the Internet Explorer installation on your agents is set-up as described below. Otherwise, you might experience issues during replay.

- In Internet Explorer, open the **Internet Options** and set the home page to `about:blank`.
- In the **Internet Options**, on the **Advanced** tab, in the **Security** section ...
  - disable **Check for publisher's certificate revocation**
  - disable **Check for server certificate revocation**
  - disable **Warn about certificate address mismatch**
- In the **Internet Options**, on the **Security** tab, do the following for each zone (Internet, Local intranet, Trusted sites, Restricted sites):
  - uncheck **Enable Protected Mode**
  - set the lowest possible security level
- In Internet Explorer, in the **Compatibility View Settings** ...
  - disable **Display intranet sites in Compatibility View**
  - disable **Use Microsoft compatibility lists**

    - remove all added websites
- On Windows Server operating systems, open the **Server Manager** and disable the **IE Enhanced Security Configuration** (IE ESC).

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.

Testing Apps for Mobile Devices

Mobile Apps Overview

Due to smaller screen size and different input methods (touch screen) of mobile devices, many Web applications look different when loaded on a mobile device compared to a browser on a PC. From a load testing perspective such applications should be treated as two separate applications, even though they might share some components on the back-end.

Silk Performer fully supports performance testing of mobile Web and native apps. It offers simulation capabilities for a variety of mobile devices, such as iPhone, iPad, Android, Windows Phone, and Blackberry, but other mobile browsers can also be simulated with custom profiles.

An interesting and important simulation parameter for mobile app testing is the bandwidth limitation. This option is not significant for usual Web testing via PCs due to the fast wired (broadband) internet connections, however for real world simulation of mobile devices it makes a real difference to set the bandwidth to particular limits. Silk Performer offers many standard bandwidth limitation settings used in the mobile world, such as EDGE, HSDPA, or LTE.

Recording Mobile Apps Using Mobile Devices

Things to consider when recording via a mobile device:

- Disable security warnings in the browser of your mobile device
- Disable "String completion" in the browser of your mobile device, otherwise a Web call will be scripted after each key press
- Delete your browser's cookies before attempting to record a script
- Operating system-specific instructions for your specific device are easy to find on the internet using your favorite search engine

To test mobile device apps, the recommended option is to record apps directly via your mobile device if it supports a configurable proxy.

1. Click **Start Here** on the workflow bar. The **Outline Project** dialog box appears.
2. In the **Name** text box, enter a name for your project.
3. Enter an optional project description in **Description**.
4. In the **Type** menu tree, select **Web Browser** > **Mobile Devices**.
5. Click **Next** to create a project based on your settings.
6. On the **Workflow - Model Script** dialog box, click **Record via mobile device**. The **Record via mobile device** dialog box appears.
7. On your mobile device, configure the browser to use the IP address and port of the machine where Silk Performer is installed as proxy. Enter the information as it is displayed in the table on the **Record via mobile device** dialog box.

   Note: Configuring your mobile device's browser proxy settings depends on the device and operating system that you are using. Instructions for your specific device are easy to find on the internet using your favorite search engine.
8. When you are done with configuring your mobile device's browser proxy settings: If you have already configured a certificate or if you do not need to record over a secure connection, click **Start recording** and skip the following step. If you want to record over a secure connection and you have not yet configured the Micro Focus CA (certificate authority) certificate or the certificate of your system under test, click **Record over a secure connection** on the **Workflow - Model Script** dialog box and proceed with the following step.
9. On the **Configure secure recording** dialog box, select if you want to use the Micro Focus CA (certificate authority) certificate or if you want to use the certificate of your system under test. If you are unsure which approach to use, refer to *Secure Connections and Certificates*.

   If you selected **Configure Micro Focus certificate**:

   a) The Silk Performer Recorder dialog opens in minimized form and the **Record a secure connection with the Micro Focus CA certificate** dialog box appears.
   b) Scan the QR code and install the certificate on your mobile device.
   c) When you are done, click **OK** on the **Record a secure connection with the Micro Focus CA certificate** dialog box.

   If you selected **Configure server certificate**:

   a) In the **Server certificate** field, locate the server certificate you want to use.
   b) In the **Pass phrase** field, enter the pass phrase that is to be used, if the server certificate you want the Recorder to use is protected with a pass phrase.
   c) *Optional:* Check the **Send root CA during SSL handshake** check box to have the Recorder send the root CA certificate during the SSL handshake. The root CA certificate may be requested by a client to authenticate the certificate authority that signed the Recorder server certificate.
   d) *Optional:* In the **Root CA certificate** field, locate the root CA certificate that you want to use.
   e) After you have completed configuring the server certificate, click **Start recording**.
10. Now interact with the Web browser or the native app on your mobile device to record a script.
11. When you are done, click **Stop Recording** in Silk Performer Recorder and save the recorded script.

Recording Mobile Apps With a Browser

1. Click **Start Here** on the workflow bar. The **Outline Project** dialog box appears.
2. In the **Name** text box, enter a name for your project.
3. Enter an optional project description in **Description**.
4. In the **Type** menu tree, select **Web Browser** > **Mobile Devices**.
5. Click **Next** to create a project based on your settings.
6. On the **Workflow - Model Script** dialog box, type the URL of the application under test in the **URL** text box. Option: Click **Analyze web page** to find visual breakpoints for your web application.
7. Click **Certificate...** if you want to use the certificate of your system under test. If you are unsure which approach to use, refer to *Secure Connections and Certificates*.
8. Click **record using customized browser**. The **Record an application using a customized browser** dialog box appears.
9. Select a **Simulation Browser**.

   Due to the multitude of technologies that browser-based applications are based on, different recording types are available for selection.

   Note: Click **Settings** to change the recording profile used for recording in Silk Performer profile settings (**Settings** > **System** > **Recorder** > **Recording Profiles**).
10. Select a predefined mobile device or a custom resolution and click **Start recording**.
11. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.
12. To end recording, click the **Stop Recording** button.
13. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.

Recording Mobile Apps With a Mobile Emulator

Recording mobile apps with an emulator follows the standard workflow, however you need to set up and configure an operating system-specific emulator on your computer where you record your test. Example procedures can be found in the community wiki:

- *Recording Mobile Apps on the Android Emulator with Silk Performer*
- *Mobile App Recording with Silk Performer and the Windows Phone Emulator*

Replaying Apps for Mobile Devices

Simulate transactions as virtual mobile device users to test applications for mobile devices.

1. In the menu bar, click **Settings** > **Active Profile** . The **Profile** dialog box appears.
2. Click **Replay** and **Internet**. In the **Bandwidth Simulation** section, define the required bandwidth configuration. For mobile traffic simulation one of the following might be appropriate:
   - GPRS
   - EDGE
   - UMTS
   - HSDPA
   - HSPA+
   - LTE
3. Click **Web** to select a mobile device. The following mobile device settings are available:
   - iPhone

- iPad
- Android
- Windows Phone
- Blackberry

Note: If you need a different mobile browser type, you can use the `Custom` option to specify your browser's parameters.

4. Follow the remaining workflow steps to define your workload and run the test.

Responsive Web Design Testing

Responsive web design (RWD) is a design paradigm for web applications that allows to dynamically adapt the user interface to the physical conditions of the end user device.

In other words: A web application renders its user interface differently depending on the viewport of the browser. For example, a web page with a three column layout on a desktop monitor might switch to a two column layout on a tablet and a single column layout on a mobile phone. Or the application switches from a one column layout in portrait orientation to a two column layout in landscape orientation.

Nicely applied RWD patterns also allow to optimize a web page in terms of resources. A low resolution viewport makes the application display low resolution images, whereas on a high resolution screen the same application loads a high resolution version of the embedded images.

For load testing, RWD means that testing an application from more than the one usual end user perspective becomes necessary. Beside different user groups, different browsers, and different network conditions, you also need to consider the end user device or viewport, which adds another dimension in test combinations.

BMC Remedy IT Service Management Support

The workflow for testing BMC Remedy IT Service Management with Silk Performer is the same as the workflow for testing Web-based applications, with the following exceptions:

- On the **Workflow - Outline Project** dialog, you must select the `Remedy` version under test (these settings are grouped under the **ERP/CRM** node).
- On the **Workflow - Model Script** dialog, you enter the commandline path to the Remedy installation under test.
- Recorded scripts include a number of `WebRemedy` functions, which are modified `WebPagePost` and `WebPageUrl` calls.
- In the `Forms` section of test scripts, each Web form contains an XML fragment that is a visualization of the Remedy traffic that was generated by the browser. This XML visualization of the browser traffic is easier to read than actual Remedy traffic. It also facilitates Silk Performer script customization. The actual, originally recorded Remedy traffic is commented at the end of each Web form XML fragment.

BMC Tutorial

BMC has created an excellent tutorial that illustrates how to load test BMC Remedy IT Service Management (ITSM) applications with BMC Remedy Action Request System (AR System). To obtain the tutorial, contact BMC and request the following white paper: *Performance Benchmarking Kit: Using Incident Management with Silk Performer* .

Licensing

A premium virtual user license is required to test Remedy with Silk Performer.

TrueLog Support

TrueLog functionality is not supported for Remedy testing.

Data Representation in Silk Performer

Remedy uses a proprietary format for data that is sent from the client to the server. This format includes length prefixes and makes customization difficult.

For example:

```
157/GetTableEntryList/11/smi-web-00126/
SMI:TEL35:SHR:AgentConsole12/Windows_View9/53687098211/
smi-web-00127/SMI:TEL35:SYS:IncomingEmail0/
1/01/02/0/0/2/0/2/0/2/0/
```

Such data is sent in HTTP requests to the URL `http://<hostname>/arsys/BackChannel`.

Remedy versions 6.3 patch level 2 and earlier send such data in the body of HTTP POST requests. Versions 6.3 and later send such data in the query string of HTTP GET requests.

The main benefit of Silk Performer's Remedy add-on is that this proprietary data format is translated into an XML-based representation that eliminates the length prefixes and introduces names to the individual data items. This makes scripts easier to understand and customize. During script replay, this XML-based format is transparently translated back into the original format.

This backward transformation is performed by the functions `WebRemedyBackChannelUrl` and `WebRemedyBackChannelPost`.

`WebRemedyBackChannelUrl` is a replacement for the function `WebPageUrl`, while `WebRemedyBackChannelPost` is a replacement for the function `WebPagePost`.

For example, without the Remedy add-on the following API call would be recorded as:

```
WebPagePost("http://213.131.176.85/arsys/BackChannel",
    "157/GetTableEntryList/11/smi-web-00126/
SMI:TEL35:SHR:AgentConsole12/W"
    "indows_View9/53687098211/smi-web-00127/
SMI:TEL35:SYS:IncomingEmail0/1"
    "/01/02/0/0/2/0/2/0/2/0/",
    STRING_COMPLETE,
    "text/plain; charset=UTF-8",
    "arsys/BackChannel");
```

With the Remedy add-on the API call is recorded as follows:

```
WebRemedyBackChannelPost("http://213.131.176.85/arsys/BackChannel",
"<?xml version='1.0'?>\r\n"
"<SegueRemedyXml
operation=\"GetTableEntryList\">\r\n"
"  <string name=\"table_server\">smi-web-001</string>\r\n"
"  <string name=\"table_schema\">SMI:TEL35:SHR:AgentConsole</string>\r\n"
"  <string name=\"table_vui_name\">Windows_View</string>\r\n"
"  <long name=\"table_field_id\">536870982</long>\r\n"
      "  <string name=\"server\">smi-web-001</string>\r\n"
"  <string name=\"schema\">SMI:TEL35:SYS:IncomingEmail</string>\r\n"
      "  <string name=\"app_name\"></string>\r\n"
      "  <long name=\"start_row\">0</long>\r\n"
      "  <long name=\"num_rows\">0</long>\r\n"
"  <Array name=\"sort_order\"></Array>\r\n"
      "  <string name=\"qualification\"></string>\r\n"
"  <Array name=\"qual_field_ids\"></Array>\r\n"
      "  <Array name=\"qual_field_values\"></Array>\r\n"
      "  <Array name=\"qual_field_types\"></Array>\r\n"
"</SegueRemedyXml>",
      STRING_COMPLETE,
"text/plain; charset=UTF-8",
      "BackChannel - GetTableEntryList");
```

The key differences are:

- Use of the function `WebRemedyBackChannelPost` instead of `WebPagePost`.
- Data is in XML format instead of the original format.
- XML format does not contain length prefixes, which would otherwise have to be considered during script customization.
- XML format specifies data type and meaningful names for each data item.

Additional Remedy Add-On Benefits

In addition to the different representation of the data there are other benefits to using the Remedy add-on:

- `Remedy.bdh`: A bdh file that implements the functions `WebRemedyBackChannelPost`, `WebRemedyBackChannelUrl` and `WebRemedyInit`.
- The recorder places a call to the function `WebRemedyInit()` in the `TInit` transaction.
- The function `WebRemedyInit()` installs verifications to catch application-level errors.
- The recorder detects all timestamps which would otherwise be hard coded into the script, and records the function `GetTimeStamp()` instead.
- The recorder records parsing functions for IDs of newly created entities and appropriate substitutions for all occurrences of such IDs in the script with the parsed variable.

Remedy SilkEssential

The Remedy application type is triggered through a SilkEssential package. It contains a set of recording rules and a `remedy.bdh`. The `remedy.bdh` wraps the `WebPageCalls`:

```
// wrapper function for WebPagePost
// param nDataLength will be ignored
function WebRemedyBackChannelPost(sUrl        : string;
                                  sData       : string;
                                  nDataLength : number optional;
                                  sContent    : string optional;
                                  sTimer      : string optional;
                                  formUrl     : form optional) : boolean
<API_FUNCTION>
  begin
    WebRemedyBackChannelPost := WebPagePost(sUrl,
RemedyBackChannelData(sData),
    STRING_COMPLETE, sContent, sTimer, formUrl);
  end WebRemedyBackChannelPost;
```

and

```
// wrapper function for WebPageUrl for use with BackChannel requests
  function WebRemedyBackChannelUrl(sUrl :    string;
                                   sTimer :  string optional;
formUrl : form optional) : boolean <API_FUNCTION>
  var
sXmlData : string(100000);
sDummy   : string;
begin
WebFormExpand(formUrl, sDummy, 1, true);
WebFormValueGet(formUrl, sXmlData, sizeof(sXmlData), "param");
WebFormValueSet(ARSYS_BACKCHANNEL_INTERNAL_HELPER, "param",
RemedyBackChannelData(sXmlData));
WebRemedyBackChannelUrl := WebPageUrl(sUrl, sTimer,
ARSYS_BACKCHANNEL_INTERNAL_HELPER);
end WebRemedyBackChannelUrl;
dclform
ARSYS_BACKCHANNEL_INTERNAL_HELPER <ENCODE_URICOMPONENT> :
    "param"                     := "";
```

Protocol Detection and Conversion

Remedy protocol detection is not available by default. However, it is already built into the Silk Performer recorder and is enabled by choosing the appropriate Remedy application type when creating a new project.

The Remedy data conversion into XML and back out of XML is implemented in the `ProxyEngine` for recording and in the `perfRemedy.dll`, which is loaded into the `perfRun` by the `Remedy.bdh`.

Citrix XenApp Support

Silk Performer provides record and replay support for the testing of applications that are hosted via Citrix XenApp session and application virtualization technologies.

Citrix facilitates real-time access to shared applications over networks and the Internet. Remote access to Citrix-enabled applications can be over DSL, T1, ISDN, or dial-up. Citrix enables multiple users to run shared applications simultaneously. Communication between Citrix plug-ins (clients) and servers consists of exchange of user inputs (keyboard/mouse) and screen shots.

Silk Performer supports both Citrix server recording and Citrix hosted application recording. Citrix plug-ins offer access to specific applications and multiple desktop windows.

The Citrix Web Interface enables users to access hosted XenApp applications and XenDesktop virtual desktops without connecting directly to host computers. Users access these resources either via a Web browser or the Citrix online plug-in.

Silk Performer offers two Citrix project types:

- *Citrix* - Used to test hosted applications that are accessed via Citrix plug-ins (clients)
- *Citrix Web Interface* - Used to test Citrix-enabled applications that are published via Citrix Web interface

The PDF-based Citrix XenApp Tutorial walks you through the entire process of testing Citrix-enabled applications. The tutorial is available in Silk Performer at **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Documentation** > **Tutorials** > **Citrix XenApp** .

Note: Silk Performer Help includes all of the content that is included in the Citrix XenApp Tutorial.

Citrix Program Neighborhood

Rather than relying on the Web Interface to distribute applications, earlier versions of Citrix use the Windows-based Citrix Program Neighborhood. The Citrix Program Neighborhood allows users to view a listing of all available applications that are by Citrix XenApp on-demand application delivery servers. An enumeration of available resources takes place automatically each time you launch the Citrix Program Neighborhood via the Citrix Browser Service.

Important: When applications are distributed via the Citrix Program Neighborhood you must configure the Citrix published application recording service to start on your system or the Citrix Program Neighborhood will not be able to update your applications list.

Defining Projects

The first step in conducting a Citrix load test is to define the basic settings for your Silk Performer project. Following that you need to perform prerequisite XenApp server configurations.

Configuring the Citrix client software

Before defining a Citrix project you must install the Citrix client software (Citrix Receiver) and configure how the Citrix server is to handle time-outs. To download the Citrix client software, navigate to *http://www.citrix.com*.

Configuring the Citrix server depends on the version you are using. Following is the procedure for configuring Citrix Server 4.0.

Note: To configure Citrix Server version 4.5 or higher, go to **Start** > **Programs** > **Administrative Tools** > **Terminal Services Configuration** and double-click the **ica-tcp** option.

1. Go to **Start** > **Programs** > **Citrix** > **Administration Tools** and launch the **Citrix Connection Configuration Tool**.
2. Double-click **ica-tcp** connection.
3. On the following dialog box, click **Advanced**. The **Advanced Connection Settings** dialog box appears.
4. Set the **Disconnection timeout** limit. 1 minute is the recommended timeout.
5. Set the **Idle timeout** limit. 5 minutes is the recommended timeout.
6. Select **reset** from the **On a broken or timed-out connection** drop list.

   If this option is not selected, sessions will remain open after replay stops due to broken or timed-out connections. This will generate replay problems when users next log into sessions as scripts will continue from the point where they left off in the previous session. In such instances sessions need to be ended manually.

Defining Your Citrix XenApp 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 **Terminal Services** > **Citrix** or **Terminal Services** > **Citrix Web Interface**.

   The **Citrix** application type uses the Silk Performer Citrix Recorder to test the delivery of Citrix-enabled applications that are accessed via any Citrix online or offline plug-in.

   The **Citrix Web Interface** application type is used to test the delivery of Citrix-enabled applications that are accessed via the Citrix Web Interface.
5. Click **Next** to create a project based on your settings.

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

Creating a Citrix Plug-In Test Script

The easiest approach to recording user actions via Citrix XenApp plug-ins (clients) that connect directly to Citrix XenApp servers and then creating a test script is to use the Silk Performer Recorder, the Silk Performer engine used for capturing and recording traffic and generating test scripts.

The Silk Performer Recorder captures and records the traffic between Citrix XenApp plug-ins and the server under test. When recording is complete, the Silk Performer Recorder automatically generates a test script based on the recorded traffic. Scripts are written in the Silk Performer scripting language, Benchmark Description Language (BDL).

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **Recording Profile** list, select `Silk Performer Citrix Recorder` to record a Citrix XenApp-enabled application via the Silk Performer Citrix Recorder application.

   This recording profile is appropriate for testing Citrix plug-ins that connect directly to Citrix XenApp servers.
3. *Optional:* If you have an ICA file that defines your server connection parameters, type the full path of the ICA file into the **Command line** field.
4. Click **Start recording**.

5. The Silk Performer Recorder then opens in minimized form along with the Silk Performer Citrix Recorder.

   If correct login credentials are not available at startup, the **Connect** dialog will open. The **Connect** dialog is also accessible via the **Connect** button in the upper left corner of the Silk Performer Citrix Recorder.

   Note: If you have an ICA file that defines your server connection parameters, select **ICA File** and browse to the ICA file to skip past all of the **Login** fields.
6. On the **Connect** dialog in the **Server** field, enter the name of the Citrix XenApp server that is to be recorded.
7. Complete the **User name**, **Password**, and **Domain** fields for the server under test.
8. Type the name of the hosted application in the **Application** field.
9. The **Client name** field enables you to specify a client name for your session. The default client name for the Citrix recorder is `SP_Recorder` (the default client name for the Citrix Player is `SP_User_x`). If you specify a different client name for recording, then a `CitrixSetClientName` function will be inserted into the script. In such cases you must customize the client name value, otherwise all users will use the same client name, which may lead to replay problems.
10. Select the desired **Color depth** for recording.
11. Select the desired screen **Resolution** for recording.

    Note: Do not change the resolution after recording your script, as replaying with a different resolution setting may cause mouse actions to fail on repositioned elements.
12. Click **Connect** to begin the Citrix XenApp session.
13. Interact with the shared desktop in the Citrix Recorder in the same way that you want your virtual users to act during the test. For example, you can click links, open applications, and enter data. Your actions will be captured and recorded by the Citrix Recorder.

    The Citrix Recorder supports session sharing, allowing you to start additional published applications in the existing session (by clicking **Run Application** on the Citrix Recorder toolbar). Click **Select Window** to switch between applications.

    The following example Citrix XenApp session includes simple Excel calculations in which the mouse and keyboard are used to open Excel, enter new data values, insert an AutoSum formula, select a screen region, edit the AutoSum formula, and close Excel.
14. When you are done recording your Citrix XenApp session, click **Stop** on the Silk Performer Citrix Recorder. The **Generating script** progress window opens followed by the **Save As** dialog.
15. Save the script with a meaningful name.
16. A newly generated test script that is based on your recorded actions appears in the Silk Performer script editor window.

Citrix Web Interface Sessions (NFuse)

Citrix Web Interface software (previously known as NFuse) provides Web access to Java, UNIX, and Windows applications that are hosted via Citrix application server software. While Citrix offers server-side control of hosted applications, Citrix Web Interface makes applications accessible through a Web browser interface (Internet Explorer, version 5.5. or higher).

For technical support and questions regarding Citrix Web Interface, go to *http://support.citrix.com*

Defining a Citrix Web Interface 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. In the **Type** menu tree, select **Terminal Services** > **Citrix Web Interface**.
5. Click **Next** to create a project based on your settings.

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

Creating a Citrix Web Interface Test Script

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

The Silk Performer Recorder captures and records traffic between a Citrix Web Interface client application (Internet Explorer, version 5.5 or higher) and the server under test. When recording is complete, the Silk Performer Recorder automatically generates a test script based on the recorded traffic. Scripts are written in the Silk Performer scripting language, Benchmark Description Language (BDL).

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **Recording Profile** list, select `Citrix Web Interface`.

   The `Citrix Web Interface` recording profile is only appropriate for testing Citrix Web Interface/ NFuse sessions.
3. Click **Start recording**.
4. The Silk Performer Recorder then opens in minimized form along with Internet Explorer. Enter the name of the Citrix server in the Internet Explorer **Address** field and click **Enter**. To see a report of the actions that occur during recording, maximize the Silk Performer Recorder dialog by clicking **Change GUI size** on the Recorder toolbar.
5. To log into the Citrix Web Interface, enter your **Username**, **Password**, and **Domain** into the Citrix Web Interface login screen. Contact your system administrator if you do not have this information.
6. Click **Log In**.
7. The application portal appears. This portal contains the applications that have been published for shared use. Select the hosted application you want to record.
8. The hosted application appears in the Silk Performer Citrix Recorder. Interact with the shared application in the Citrix Recorder in the same way that you want your virtual users to behave during the test. Your actions will be captured by the Citrix Recorder and generated into a BDL script.
9. When you close the application the Citrix session disconnects and you can save your recorded script.

BDL scripts of recorded Citrix Web Interface sessions are multi-protocol scripts that include a small number of Silk Performer Web functions.

Citrix Script Functions

- `CitrixWaitForWindowCreation`, used for screen synchronization, is the most important Citrix function.

  The first parameter that synchronizations consider is window caption. If during replay a window caption is returned that matches the caption of the window that was recorded during replay, then the verification succeeds.

  Note: Wildcards (*) can be specified at the end or beginning of window captions.

  The second parameter considered is the match parameter. Normally this is an exact, case-sensitive comparison of window caption strings.

  The third parameter is the style of the window. If a window caption name is not available, then a portion of the style name is used to identify the window. Styles reflect whether or not windows were maximized during recording.

The fourth parameter considered is the position of the window. This may be a negative number, as maximized windows have a position of `-4, -4`. This parameter can be controlled via the **Force window position** profile setting. When this profile setting is enabled, windows are automatically moved to the exact positions they held during recording.

The fifth parameter is window size. During replay windows must be resized to the same size they had during recording.

- `CitrixWaitForLogon` waits until a Citrix client logs on successfully or a specified timeout period expires.
- `CitrixWaitForScreen` captures screen regions and checks them against specified conditions (normally a hash value captured at recording). If a condition does not match and the timeout period expires, the function call fails. `CitrixWaitForScreen` functions also use screen appearance to synchronize subsequent user actions. Based on provided parameters, this function waits until the image in a specified screen region changes, matches the hash value captured at recording, or does not match the hash value captured at recording.
- `CitrixGetScreen` takes a screenshot of a selected region and writes the screenshot to a file in the result directory. If the file name is omitted, it will be automatically generated by the user ID and hash value of the image.
- `CitrixGetScreenHash` retrieves the hash value of a selected screen.
- `CitrixSetOption` sets particular options, such as network protocol specification, speed screen latency reduction, data compression, image caching, mouse/keyboard timings and event queueing, client disconnect, synchronization time-outs, think times, TrueLog capture, and window position forcing.
- `CitrixWaitForWindow` waits until a specified window event (specified with the `nEvent` parameter) occurs. Such events may be an activation, destruction or caption change for a specified window. If the selected event is a caption change, a matching caption change for the specified window will satisfy the function. Captions can be specified explicitly using the `sCaption` and `nMatch` parameters.
- `CitrixKeyString` is the standard function for entering printable characters.
- `CitrixMouseClick` moves the mouse to a specified position and presses a specified button. Optionally, the mouse can be specified to move while the button is pressed. Key modifiers (such as `Ctrl` and `Alt`) can be used.

Screen Synchronization and Verification

Silk Performer supports bitmap and window verification for applications that are hosted by Citrix XenApp servers. Screen synchronization offers a means of verifying replayed Citrix content. Screen synchronizations differ from standard script verifications in that they allow for verification of window and screen region appearances, not input values. Also, they are inserted during script recording, not during subsequent script customization. Screen synchronizations are particularly useful for synchronizing subsequent user input that is displayed in browsers, or similar interfaces (for example, Citrix application windows), because such applications do not support automatic synchronization through window events such as user input and text display.

Note: Response data verification is not supported for Citrix testing.

Silk Performer relies on hash values to verify replayed bitmaps against recorded bitmaps. Hash values are computer-readable values that reflect bitmap specifications such as size, position, resolution, and color depth.

Note: To verify replay screen regions against hash values that are captured at recording it is necessary that the same color depth that is used during recording be used during replay. Scripts fail when these specifications are not maintained because changes as small as a single pixel can change hash values and result in replay content appearing to be different from recorded content.

Note: Windows maximized during recording must also be maximized during replay. This is because replay cannot change the state of windows (it can only resize and move windows). So if a window state changes (for example, from `Maximized` to `Restored`), then it is likely that some user input in the script caused the change (for example the **Restore** button may have been clicked). On replay the user will click at the same screen position (now the **Maximize** button) and consequently a different operation will be executed and the subsequent `CitrixWaitForWindowRestore` function will fail.

Text and Screen Synchronizations

Window synchronizations such as `CitrixWaitForWindow()` and `CitrixWaitForWindowCreation()` are well suited to synchronizing with an application. It is important to synchronize with the application so that the script waits until the application is ready for additional user input. If you do not use synchronizations, the script will most likely overrun the application. Also, synchronization gives you point-in-time reference as to when tasks are completed, effectively measuring response times.

Text Synchronization

Many tasks that can be performed on an application do not show, hide, create, or destroy windows, they simply change a section of the screen. In such instances you should use the Silk Performer text synchronization functionality.

After recording your script, you can visually add text synchronization points via the TrueLog Explorer `Synchronize Text` option. Text synchronization works via built-in OCR (Optical Character Recognition).

Screen Synchronization

Silk Performer offers two types of screen synchronizations: `wait for content change` and `wait for content match`. This form of synchronization requires a window reference and two sets of x/y coordinates. However since unplanned events can lead to displaced window coordinates, you should use text synchronization whenever possible, and only use screen synchronization when there is no text to compare.

Wait for content change waits until the selected portion of the screen changes, while wait for content match waits until the selected portion of the screen matches what was defined during recording.

Note: Screen synchronization points must be inserted while you record your test case, whereas text synchronization points can be inserted afterward, via TrueLog Explorer.

Generating a Screen Region Synchronization During Recording

Screen synchronization is achieved via `CitrixWaitForScreen` functions, which are not scripted automatically by the recorder. These functions are inserted via the **Screen Region** dialog box during recording. `CitrixWaitForScreen` functions compare replay and record bitmaps to determine whether or not they are identical. Hash values, as opposed to actual bitmaps, are used to compare the images. This limits resource consumption during replay.

Note: Screen region synchronization is only available via the Silk Performer Citrix Recorder.

1. Record a Citrix session and create a test script.
2. During recording, click **Select Region** on the Silk Performer Citrix Recorder.
3. Click and drag your cursor to select the screen region for which you want to generate a bitmap synchronization.

   Note: Because differences as small as a single pixel can cause synchronization processes to fail, it is recommended that for text verifications you select the minimum screen area required. Otherwise unanticipated screen differences (for example, disabled toolbars) may affect verification results.
4. The **Selection** dialog box opens. Specify how you want to have screen region coordinates scripted (`Script absolute coordinates`, `Script coordinates relative to window`, or `No coordinates, use full window`). When windows are maximized there is effectively no difference between absolute and relative coordinates. When windows are not maximized, relative coordinates are

measured from the top-left corner of the Citrix Recorder window, while absolute coordinates use fixed x/y coordinates.

5. Specify the **Content matching type** that the Citrix XenApp player should wait for during replay (`content match`, `content mismatch`, or `content change`).
6. Click **OK** to add the synchronization to your Citrix XenApp test script.

Verification and Parsing via OCR

Silk Performer support for optical character recognition (OCR) simplifies session-dependent verifications and parsing by recognizing text values in the screengrabs of captured application states.

Verification and parsing functions are added via TrueLog Explorer after script recording.

Window Position and State

Window position and state (maximized/minimized) is important for insuring accurate replay as TrueLog Explorer scripts screen coordinates where selected text is to be read from relative to the desktop, not individual windows. So if a window appears in a different position during replay than it did during recording, OCR operations will not be able to locate the specified text. If it is not possible to specify an absolute position, the script must be manually updated using coordinates relative to windows.

Adding OCR Verification Functions

With OCR support Silk Performer enables the storing of recognizable text values in variables, thereby simplifying session-dependent verifications in Citrix tests.

Overview

String verification via optical character recognition (OCR) is achieved using `CitrixVerifyText` API calls. These functions are inserted via TrueLog Explorer during script customization. `CitrixVerifyText` functions compare text strings in replay bitmaps to determine if they are identical.

`CitrixParseText` functions are available for parsing text. These API calls work in the same way as other Silk Performer parsing functions.

Optical character recognition relies on pattern databases to recognize varying fonts and text styles. Font databases must be generated before OCR can be run.

Citrix TrueLogs show verification and parsing API calls in the tree view.

Note: OCR operations must be performed on stable content because when working with frequently changing screen images replay timing is critical. When synchronizing on window events it is possible that screen refresh may be slightly delayed, which will result in timing dependent outcome. Therefore it is good practice to either script a wait or a `CitrixWaitForScreen` function call before each OCR verification/ parsing function.

The following two screen examples show the output of verification and parsing functions after TryScript runs.

Generating an OCR Verification Function

1. From Silk Performer record a Citrix session
2. Run a TryScript run with the **Animation** checkbox selected on the **TryScript** dialog box. This opens TrueLog Explorer.
3. When the TryScript run is complete, select a `CitrixSynchronization` API node (or child node) that includes a bitmap screengrab of a page on which you want to verify text.
4. Click and drag your cursor onscreen to select the page region that includes the text you want to use for verification.

5. Right-click in the selected area and select **Verify Selected Text** from the context menu.
6. The **Insert Text Verification Function** dialog box opens. The selected text is pre-loaded into the constant value edit box and the constant value radio button is selected by default.

   In addition to being able to verify against a constant value, you can also verify against an existing parameter or a new parameter. To verify against a parameter, select the parameter radio button. If a parameter already exists, clicking "**...**" enables you to browse to and select the parameter. If no parameters exist, clicking "**...**" launches the **Parameter Wizard**, which you can use to create a new parameter.
7. From the **Verify that the text in the selected rectangle is** drop list, select `equal` or `not equal`.
8. Specify whether or not the verification is to be **Case sensitive** and should **Ignore whitespaces**.
9. In the **Severity** portion of the dialog box, specify the severity that is to be raised if the verification returns a negative result (`Error`, `Warning`, `Informational`, or `Custom`).
10. Click **OK**.
11. A confirmation dialog box appears. Click **OK** to add the OCR verification function to your Citrix test script.

Generating an OCR Parsing Function

1. From Silk Performer record a Citrix session
2. Run a TryScript run with the **Animation** checkbox selected on the **TryScript** dialog box. This opens TrueLog Explorer.
3. When the TryScript run is complete, select a `CitrixSynchronization` API node (or child node) that includes a bitmap screengrab of a page from which you want to parse text.
4. Click and drag your cursor to select the page region that includes the text you want to parse.
5. Right-click in the selected region and select **Parse Selected Text into a Variable**.
6. The **Insert Parsing Function** dialog box offers parameters by which the parsing function can be configured. Though the default settings will likely be correct, you can adjust:

   Parameter name - Enter the name of the parameter that is to receive the result of the parsing function.

   Informational statement insertion - Select **Print statement** to insert an informational Print statement into the script after the Web page call. This writes the result of the parsing function to the Silk Performer **Virtual User Output** window.

   Select **Writeln statement** ("write line" statement) to write the parsed value to an output file to facilitate debugging (in addition to writing the value to the **Virtual User Output** window as a Print statement does). Because generating output files alters test measurements, these files should only be used for debugging purposes and should not be generated for full tests.
7. Click **OK**.
8. A confirmation dialog box appears. Click **OK** to add the OCR parsing function to your Citrix test script.

Trying Out a Generated Script

With TryScript 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: Although Citrix TrueLogs do not include live display of data downloaded during testing (via TrueLog Explorer) Citrix Web Interface TrueLogs do include live display of downloaded data. Both TrueLog types include the writing of log files, report files, and replay within the Silk Performer Citrix Player.

If you have configured parsing or verification functions based on Citrix OCR support, you must generate an OCR font database before attempting a TryScript run, otherwise these functions may not operate correctly.

1. Click **Try Script** on the Silk Performer Workflow bar. The **Try Script** dialog box opens.

2. To view live display of page content within TrueLog Explorer during replay (Citrix Web Interface TrueLogs only), select the **Animated Run with TrueLog Explorer** check box.

   The Visible Citrix Client option (Citrix TrueLogs only, not available for Citrix Web Interface TrueLogs) enables visual replay in the Silk Performer Citrix Player during TryScript runs.
3. Click **Run**.

   Note: You are not running an actual load test here, only a test run to see if your script requires debugging.
4. The TryScript run begins. The Silk Performer Monitor window opens, giving you detailed information about the run's progress.

Citrix TrueLogs

The Silk Performer Citrix Player open for Citrix TryScript runs. TrueLog Explorer opens for Citrix Web Interface TryScript runs (when the **Animation** checkbox on the TryScript dialog box is checked). TrueLog Explorer displays the data that is actually downloaded during TryScript runs.

By selecting a high-level synchronization node you see a bitmap of the window captured during replay just as it appeared after the last synchronization function.

Window synchronization functions are visualized with colored borders. Window creations are indicated with green borders. Window activations are indicated with blue borders. Window destructions are indicated with yellow borders.

TrueLogs work in complement with the Silk Performer Citrix Player by visualizing screen states. For example, if you are not sure which window is indicated by a certain window ID that is listed in the Silk Performer Citrix Player Log window, you can find the corresponding synchronization function in the corresponding TrueLog and thereby access a bitmap that shows the window.

User input nodes (`CitrixUserInput` and related functions) reflect keyboard and mouse input. `CitrixMouseClick` functions offer two track vector parameters (X and Y coordinates). Red diamonds indicate mouse-click start points. Red cross-marks indicate mouse release points. A red line between a start and end point indicates the path of the mouse. If there is no move while the button is pressed, then only a red cross is displayed. Onscreen tool tips offer additional information (for example, `right-click`, `left-click`, `double-click`).

Value strings (keyboard input) are visualized onscreen as floating red text until target window captions are identified (in subsequent nodes) to indicate where strings are to be input.

Silk Performer Citrix Player

The Silk Performer Citrix Player opens when TryScript runs begin, replaying all recorded actions in full animation. Mouse movements and operations are simulated with an animated mouse icon.

Note: Silk Performer Citrix Player only opens for the Citrix application type, not the CitrixWeb Interface application type.

Click **Toggle Log Window** in the upper right corner of the player to open the **Log** window. The **Log** window includes three panes that detail different aspects of TryScript runs:

- **Script** - This pane lists all of the executed BDL script functions and the currently executing BDL function.
- **Windows** - This pane includes a stack of all the client windows of the current session, including window captions, styles, sizes, and positions. Top-level windows carry a window icon and are listed above sub-windows.
- **Log** - This pane lists all informational messages and events, including executed BDL functions, and window creation, activation, and destruction.

In all panes, active functions and windows are indicated with a blue arrow icon.

Step by Step Replay

1. During a Citrix TryScript run, click the **Toggle Log Window** button in the upper right corner of the Silk Performer Citrix Player to open the **Log** window.
2. Click the **Step** button. Replay stops at the active function. A blue arrow icon indicates the next function in the script.
3. Click **Step** to execute the next function.
4. Continue clicking **Step** to advance through the script.

   Note: You can also enable step-by-step execution by selecting the **Step by step execution** checkbox on the **TryScript** dialog box.
5. Click **Run** to resume continuous script processing.

Skipping Time-Outs

The Silk Performer Citrix Player waits for all time-outs that are encountered during replay. To avoid waiting for time-outs (default time-out is 60 seconds), click **Skip** to advance past them.

Note: Clicking **Skip** generates user-break errors in scripts.

1. During a Citrix TryScript run, click the **Toggle Log Window** button in the upper right corner of the Silk Performer Citrix Player to open the **Log** window.
2. When replay encounters a time-out, click **Skip** to advance to the next function.

Customizing User Data

With user data customization you can make your test scripts more realistic by replacing static recorded user input data with dynamic, parameterized user data that changes with each transaction. Manual scripting is not required to create such data-driven tests.

During testing you can customize the user input that is entered into applications that are hosted by Citrix terminal services in two ways:

- The **Parameter Wizard** allows you to specify values to be entered with keyboard events, enabling your test scripts to be more realistic by replacing recorded user input data with randomized, parameterized user data.
- Visual customization allows you to customize mouse events such as clicks, drags, and releases.

Using the Parameter Wizard

1. Select an API node that reflects user data input (for example, select a `CitrixKeyString` node that specifies a keyboard datastring).
2. Right-click the input datastring (shown as floating red text) and select **Customize User Input** from the context menu.
3. The **Parameter Wizard** opens. Select **Create new parameter** and click **Next**.
4. With the **Parameter Wizard** you can modify script values in one of two ways. You can either use an existing parameter that's defined in the `dclparam` or `dclrand` section of your script, or you can create a new parameter (based on either a new constant value, a random variable, or values in a multi-column data file). Once you create a new parameter, that parameter is added to the existing parameters and becomes available for further customizations.

   Note: This task explains only the process of creating a parameter based on a new random variable.
5. The **Create New Parameter** dialog appears. Select the **Parameter from Random Variable** radio button and click **Next**.

6. The **Random Variable Wizard** appears. From the drop list, select the type of random variable (for example, `Strings from file`) you want to insert into your test script. A brief description of the highlighted variable type appears in the lower window.
7. Click **Next**.
8. The **Name the variable and specify its attributes** page appears. The `Strings from file` random variable type generates data strings that can either be selected randomly or sequentially from a specified file. Enter a name for the variable in the **Name** field. Specify whether the values should be called in `Random` or `Sequential` order. Then select a preconfigured datasource (for example, `Elname` which defines last names) from the **File/Name** drop list.
9. Click **Next**.
10. The **Choose the kind of usage** page appears. Specify whether the new random value should be used `Per usage`, `Per transaction`, or `Per test`.
11. Click **Finish** to modify the BDL form declaration of your test script so that it uses the random variable for the given form field in place of the recorded value. The new random variable function appears below in BDL view.
12. Initiate a TryScript run with the random variable function in your test script to confirm that the script runs without error.

Customizing Mouse Events

1. Select a `CitrixMouseClick` node that includes mouse activity. Red diamonds indicate mouse-click start points. Red cross-marks indicate mouse-release points. A red line between a start and end point indicates the path of the mouse. Onscreen tooltips offer additional information (for example, `right-click`, `left-click`, and `double-click`).
2. Click anywhere on the screen and select **Customize User Input** from the context menu. The **Customize Mouse Event** dialog box appears.
3. Click at the screen position where you want the customized mouse move to begin.
4. Click at the screen position where you want the customized mouse move to end.
5. Click the **Customize** button to accept the customization and modify the BDL script accordingly.

   Your mouse event customization now appears in the recorded TrueLog bitmaps in green. The mouse customization also appears in the BDL script in green text. `CitrixMouseClick` functions offer two track vector parameters (X and Y coordinates). The next time this script executes, it will use the new screen coordinates you have specified.

Synchronizing Text

TrueLog Explorer offers a synchronization function that pauses the execution of Citrix functions until specified text or a text pattern appears in a specified location. Such synchronization is vital for the accurate execution of verification functions.

Project and System Settings

Citrix profile settings are project-specific settings related to Citrix synchronization, logging, virtual user simulation, and client options. Citrix settings are specified on a per-project basis.

This section focuses on Citrix replay settings. Citrix record options are limited to network protocol, encryption level, the `Log screen before each user action` setting, and the `Use RAM disk` setting.

Configuring Citrix XenApp Options

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.

3. In the shortcut list, scroll down to and click the **Citrix** icon. The **General** tab opens.
4. In the **Synchronization timeout** text box, type a default synchronization timeout in milliseconds that is to be used by all `CitrixWaitForXXX` functions that do not specify a timeout value.

   The default value is `60000 ms`.
5. Check **Force window position** to make the calls to `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` move windows to the coordinates captured during recording (enabled by default).

   When this check box is not checked, both the `CitrixWaitForWindowCreation` and `CitrixWaitForWindowRestore` functions provide the parameter `bForcePos` to enable this option for each individual call.
6. Check **Disconnect on transaction end** to disconnect the Citrix client following the end of each transaction, including the `TInit` transaction (disabled by default).
7. Check **Gracefully disconnect session** to perform a log-off when disconnecting from a Citrix XenApp session (enabled by default).
8. Check **Log screen before each user action** to capture and write a screenshot at the beginning of each user action (enabled by default).

   When TrueLog generation is enabled, a screenshot is taken at the end of each synchronization function and written to the TrueLog.

   The `CitrixWaitForScreen` function captures a screen region and checks it against a specified condition instead of against a hash value that is captured at recording. This file can later be examined or compared to what was captured during recording for error analysis.

   If the function call `CitrixWaitForScreen` fails and **Dump window region on unsuccessful screen synchronization** is checked , the captured screen region is written to a file in the result directory. The function call `CitrixWaitForScreen` may fail, for example, if the condition does not match when the timeout period expires.
9. Check **Use RAM disk** to use a different drive for the intermediate storage of images.

   Note: If you check **Use RAM disk**, you must select the appropriate drive letter of the RAM disk from the list box. TrueLog generation performance improves if the specified drive is a RAM disk.
10. Select the **Simulation** tab.
11. In the **Length of time mouse button remains pressed** field, type the length of time that the virtual user is to hold the mouse button in the pressed state.

    The default value is `200 ms`. The functions `CitrixMouseClick` and `CitrixMouseDblClick` use this value.
12. In the **Length of time between the clicks of a double-click** field, type the maximum length of time that can pass between the two clicks of a double-click.

    The default value is `100 ms`. The function `CitrixMouseDblClick` uses this value.
13. In the **Mouse speed** field, type the speed (in pixels per second) at which the mouse is to move across the screen.

    The default value is `1000 pixels`.
14. In the **Length of time each key remains pressed** field, type the length of time that the virtual user must hold a keyboard key in the down state.

    The default value is `50 ms`. The functions `CitrixKey` and `CitrixKeyString` use this value.
15. In the **Length of time between keystrokes when entering strings** field, type the length of time that must pass between the individual keystrokes.

    The default value is `100 ms`. The function `CitrixKeyString` uses this value.

16. In the **Key repeat time** field, type the time required for a complete key stroke when simulating repeat functionality.

    The default value is `50 ms`. A value of `50 ms` represents 20 keys per second.
17. In the **Delay after successful synchronization** field in the **Think times** area of the tab, type the length of time that virtual users are to remain inactive after passing a successful synchronization point.

    The default value is `1000 ms`. The function `CitrixWaitForXXX` uses this value.
18. In the **Delay after each user action** field, type the length of time that virtual users remain inactive between actions.

    The default value is `100 ms`.
19. Click the **Citrix client** tab to specify Citrix client options.
20. From the **Network protocol** drop box, select the low-level network protocol to use for locating and connecting to the Citrix server.

    For more information, refer to Citrix client documentation.
21. Check the **Use data compression** check box to compress all transferred data (enabled by default).

    This feature reduces file size but requires additional processor resources.
22. Check the **Use disk cache for bitmaps** check box to store commonly used graphical objects, such as bitmaps, in a local disk cache (disabled by default).
23. Check the **Queue mouse movements and keystrokes** check box to queue mouse and keyboard updates (disabled by default).

    This feature reduces the number of network packets sent from the Citrix client to the Citrix XenApp server.
24. From the **SpeedScreen latency reduction** drop box, select one of the following to enhance user experience on slower network connections:

    - For WANs and other slower connections, select `On`.
    - For LANs and other faster connections, select `Off`.
    - To turn latency reduction on and off based on the latency of the connections, select `Auto`.
25. From the **Encryption level** drop box, select a level of encryption for the ICA connection.

    The Citrix XenApp server must be configured to allow the selected encryption level or higher.

    Note: Using an encryption level other than the server default or `Basic` disables automatic logon to the Citrix XenApp server.
26. Click **OK** to save your settings.

Configuring General Settings

1. In the Project tab of the tree-view area of the main SilkPerformer window, right-click the Profiles node and select Edit Active Profile.
2. The Profile - [Profile1] - Simulation dialog opens at the Simulation tab (Replay category).
3. Scroll down to and select the Citrix icon. The Citrix General settings tab opens.
4. In the **Options** section of the **General** tab, specify a timeout value in the **Synchronization timeout** field. You may have to increase this value if your Citrix server is slow.
5. The `Force window position` option (enabled by default) automatically moves replayed windows to the coordinates specified in `CitrixWaitForWindowCreation` functions.
6. The `Disconnect on transaction end` option (disabled by default) disconnects the client after each transaction, even init transactions.
7. In the **Logging** section of the **General** tab, the `Log screen before each user action` option (enabled by default) enables onscreen display of user input (datastring values) for the moment before values are actually input into screens (for example, a user might enter a string value into a spreadsheet cell. The value is not actually input until the user pushes **Enter**. With this option enabled, in the node just preceding the click of the **Enter** key, the string value appears onscreen as floating red text). This

option requires significant processing and disk storage as it dictates that each user action generate a screengrab. With this option disabled you do not see all user input updates.

8. The `Dump window region on unsuccessful screen synchronization` option specifies that screengrabs be generated for all unsuccessful screen synchronizations. These bitmaps, when captured and saved to the current result directory (for example, `RecentTryScript`) of your Silk Performer installation, can be compared to corresponding recorded synchronizations to assist in debugging efforts. For example, a difference of a single pixel is enough to cause a screen synchronization error. Such a difference might best be detectable visually, by comparing recorded screengrabs with screengrabs captured when synchronization errors occur.
9. The grabbing, reading, compressing, and writing of screengrabs for TrueLogs involves significant processing resources and can lead to slow replay. The `Use RAM disk` option (disabled by default) enables faster TrueLog replay by making use of a RAM disk or solid state drive (SSD), rather than writing files to a conventional hard disk. Use the drop list to select the letter of a drive with high read/ write rates. Note that this option does not install a RAM disk, but you may use the RAM disk of your choice.
10. Click **OK** to save your changes, or click **Default** to restore the default settings.

Configuring Simulation Settings

1. In the **Project** menu tree, right-click the **Profiles** node and select **Edit Active Profile**.
2. The **Profile - [Profile1] - Simulation** dialog opens at the **Simulation** tab (Replay category).
3. Scroll down to and select the Citrix icon.
4. Select the **Simulation** tab.
5. In the **Mouse** section of the **Simulation** tab, specify virtual user mouse behavior (in milliseconds) such as the length of time that mouse clicks remain pressed, the length of time between the clicks of a double click, and mouse speed. Note that simulated mouse events move at constant speeds. They do not simply jump across the screen.
6. In the **Keyboard** section of the **Simulation** tab, specify virtual user keyboard behavior (in milliseconds) such as the length of time that keys remain pressed, the length of time between keystrokes when entering strings (for example, `CitrixKeyString` functions), and key repeat time (irepeat parameters of `CitrixKeyString` functions).
7. In the **Think times** section of the **Simulation** tab, specify virtual user think time behavior (in milliseconds) for delay after successful synchronizations, and delay after each user action. This is a virtual simulation of user reaction time that helps to stabilize replay.
8. Click **OK** to save your changes, or click **Default** to restore the default settings.

Configuring Client Settings

1. In the **Project** menu tree, right-click the **Profiles** node and select **Edit Active Profile**.
2. The **Profile - [Profile1] - Simulation** dialog box opens at the **Simulation** tab (Replay category).
3. Scroll down to and select the Citrix icon.
4. Select the Citrix client tab.
5. Select the network protocol upon which your client will run (`TCP/IP` or `TCP/IP + HTTP`). When you specify `TCP/IP + HTTP`, load balancing is done with the HTTP protocol, using post commands. When you specify `TCP/IP`, UDP is used. No other network protocols are supported.
6. Check the **Use data compression** checkbox to enable data compression (enabled by default).
7. Check the **Use disk cache for bitmaps** checkbox to enable the caching of bitmaps on your hard disk (disabled by default).
8. Check the **Queue mouse movements and keystrokes** checkbox to queue mouse movements and keystrokes for a specified period of time before they are sent to the server (disabled by default).

9. **SpeedScreen latency reduction** enables local echo of mouse and keyboard actions (disabled by default). **Local echo** means that you do not have to wait for round trips to the server to see the results of your input. Specify `Off`, `On`, or `Auto`.
10. Specify an **encryption level** for the client. Options include, `Use server default`, `Basic`, `128 Bit for Login Only`, `40 Bit`, `56 Bit`, and `128 Bit`.
11. Click **OK** to save your changes, or click **Default** to restore default settings.

Enabling Citrix Replay on Agents

1. From the Silk Performer menu bar, select **Tools** > **System Configuration Manager** . The **System Configuration Manager** dialog box appears.
2. *(applicable to all Citrix XenApp client versions)* Click the **Applications** tab. Click the Silk Performer icon. Enter **Account** and **Password** credentials for a user account on the agent workstation that has permission to execute Citrix sessions.
3. To record a Citrix published application that is accessed using the settings defined on the Citrix Recorder **Connect** dialog box, perform the following:
   a) Expand the **Profiles** node on the Project menu tree and right-click the active profile. Select **Edit Profile**.
   b) Scroll down to and click **Citrix**.
   c) Click the **Citrix client** tab.
   d) From the **Network protocol** list box , select `TCP/IP + HTTP`.

   To record a Citrix published application utilizing an ICA file: In the `[WFClient]` section of the ICA file, change `TcpBrowserAddress=` to `HttpBrowserAddress=`.

System Settings for OCR

To enable optical character recognition (OCR) for parsing and verification functions in TrueLog Explorer, you must generate a font database using Silk Performer system settings.

Optical character recognition relies on font (or pattern) databases to recognize fonts and text styles in bitmaps. The default set of fonts covers most scenarios, however in some situations you may want to add additional fonts or font styles. A new database should be generated whenever new fonts are added or removed from the system.

Including too many fonts in the database can slow down processing and lead to contradictory reading, so it is recommended that you only include those fonts that are used in the bitmaps from which you will be capturing text strings.

Configuring System Settings for OCR

1. Within Silk Performer, go to `Settings/System.../` and select the Citrix icon.
2. On the **OCR** tab, use the **Add >>** and **Add All** buttons to move those fonts that you wish to have used for OCR from the **System Fonts** list box to the **Chosen Fonts** list box.
3. Use the **Remove All** and **<< Remove** buttons to delete unnecessary fonts from the **Chosen Fonts** list box.
4. In the **Sizes** field, specify the font size range that should be used (for example, `8-20`).
5. Define which font styles should be included by selecting the `Italic`, `Bold`, and `Underlined` check boxes. Make sure you select the `Underlined` check box if you want to verify menu entries.
6. Click **Generate Font Database**.
7. The **Build Font Base** dialog box opens. Click **OK** to confirm that you want to replace the existing font database with a new database.

   Note: If you encounter problems while generating the font database, it is likely a data execution prevention issue. To resolve the issue, please see the section below.

8. Click **OK** on the Silk Performer System Settings dialog box to accept the changes.

Testing Best Practices

While GUI-based testing presents a number of challenges, it can be rewarding as it offers the opportunity to closely simulate a real user experience. GUI-based testing through Silk Performer involves applying load to applications by simulating the terminal services protocol.

At least two physical computers should be used to perform Citrix tests: one computer runs the terminal services environment and the application under test (AUT); the other computer runs Silk Performer and the Citrix client software.

The load generator computer simulates a large number of users by mass producing the terminal services network protocol. The terminal services server (system under test) simulates multiple MS Windows-based desktop sessions simultaneously. This server also hosts the GUI application that is placed under load.

We recommend that you place an isolated network (LAN) between the load generator and the system under test. This allows for network throughput analysis. An isolated network is also less vulnerable to external influence, thereby reducing errors and misleading network throughput results.

Test Preparation

- User Interface (UI) design - Take advantage of any opportunity to influence the UI design of the application. A solid, consistent UI allows you to create reusable sub-routines. In turn, you can write less test code, and consequently have less code to maintain.
- Test plan - A thoughtful test plan is a prerequisite of successful GUI-based load testing. In your test plan you should define your goals, objectives, test runs, critical metrics, and others.
- Use cases - Use cases are step-by-step scenarios that you plan to test. Use cases represent typical user-to-application interaction, for example starting an application, opening a project, editing settings, and others. Your use cases guide you through the recording process.
- Application expert - Even when equipped with well documented use cases you may not be able to make sense of all of an application's workflow. It is helpful to have access to someone who has expert knowledge of the application under test, for example a business process owner.
- Screen resolution - The higher the screen resolution, the higher the system requirements are per terminal session. We recommend you use a screen resolution of no more than 800x600 for record and replay. This approach allows for compatibility with older computers, which you then have the opportunity of including in your load tests.
- Remote users - If your application is to be available to remote users, consider simulating a slow network during some of your load tests. Slow network connections can have a major impact on the performance of applications, therefore additional tests with simulated network congestion may help improve your test results. You can use the network emulation functionality of Silk Performer for this purpose.

Defining Your Goals

The first thing to do when planning your load test is to define your goals. While testers often assume that their goals are clear to all involved parties, they often are not. It is important that the quality assurance team and management work toward the same goals.

When writing up goals, first consider your highest-level goals:

1. Discover the baseline performance of your application
2. Optimize/tune your application
3. Determine if the application under test is ready for production

Once you have established your high-level goals, you need to break them down into clear, measurable objectives, covering issues such as how you plan to accomplish your goals, how you plan to measure your goals, and what results will be considered acceptable.

Here is an example of how you might break down your testing goals:

Goal #1: Discover and document the baseline performance of your application

- Use Silk Performer to measure the response times of critical application functions.
- Place timer functions around window/screen synchronizations (`WaitFor` events). These measurements will show up in the final report.

Critical timers:

- Measure how much time it takes from when the **OK** button is clicked on the **Login** window until the **Welcome** window appears.
- Measure how much time it takes from when the **Query results** button is clicked until the populated list is displayed.

Goal #2: Optimize/tune the application

- Optimize/tune the application for 5 days.
- Document all changes and their impact on the performance of the application under test.

Goal #3: Determine if the application under test is ready for production

- The application is ready for production if the following condition is met: All response times are under 5 seconds (Silk Performer provides metrics for each window and screen sync).

With a list of measurable objectives, your test results define a definite point at which the application will be ready for production. This also eliminates the risk of endlessly optimizing the application as tuning it to the defined goal is adequate.

Creating Use Cases

A use case is a typical task that a user undertakes when working with the application under test. A use case must use the features of the application that require testing. It is essential that you only test features that are important and working properly. This is not the time to perform functional testing, which should already have been completed. Testing is a long process: the longer the use cases, the more time that will be required for testing.

When stepping through a use case, write down all significant screen events. For example, when entering a formula in Microsoft Excel you should document the changing of cells due to formula processing. Such events will translate into screen synchronizations that will be important during script development. Text synchronizations can be used for text-based screen synchronizations.

Document use cases to a detailed level. You need to document every mouse click, key stroke, and expected result. While this may be tedious initially, it makes things easier when you begin recording your test cases.

The following example, a test that simply locates an existing instance of Microsoft Word and opens a document, displays the level of detail that a use case should have. The square brackets (“[ ]”) indicate an event.

```
In the “Microsoft Word” window, navigate to the File menu and select Open….
[The “Open” dialog box opens]
Select ‘Test.doc.’
Click Open.
[The “Open” dialog box closes]
[The “Microsoft Word” window has focus once again]
```

Note that the exact titles of the windows are documented in the above example. This becomes important later during script development as such information is needed to keep the script in sync with the application. Once you have a well documented use case, it is easy to script the application

There are several ways to write use cases. You can use XML notation, numbered notation, or another format that you are familiar with.

XML Use Case Example

```
<Task Name=”Open a document”>
  In the “Microsoft Word” window, navigate to the File menu        and select
```

```
Open….
  [The “Open” dialog box displays]
  Select ‘Test.doc.’
  Click Open.
  [The “Open” dialog box closes]
  [The “Microsoft Word” window has focus once again]
</Task>
```

Numbered Use Case Example

```
100.01 – Open a document
  In the “Microsoft Word” window, navigate to the File menu        and select
Open….
  [The “Open” dialog box displays]
  Select ‘Test.doc.’
  Click Open.
  [The “Open” dialog box closes]
  [The “Microsoft Word” window has focus once again]
100.02
    …
```

Ensuring a Stable Environment

Before you start recording, make sure that there will be no additional changes to the UI of the application under test, at least until your tests are complete. This requires good communication with the development team. If the UI changes after you record your test scripts, your scripts will likely be unusable.

Also, make sure that you have the ability to back up and restore any databases that are used in your testing. Because changes to database content often lead to differences in UI content, for example more or different items in lists, different query results, and other, databases need to be returned to a baseline state before the start of each test cycle.

Troubleshooting

Once you have documented your use cases, recording or coding your test scripts should be a relatively simple task. With Silk Performer you have both the record/playback option and the option of coding your test scripts manually. Each approach has its advantages and disadvantages. The most efficient way to create test scripts with Silk Performer is to combine the two approaches, as follows:

1. With Silk Performer, record the scenario you have outlined in your use case description.
2. Use TrueLog Explorer to visually customize your script (this is where the “Text synchronization” feature comes into play).
3. For more complex scripting, edit the generated BDL script manually.

   Manually editing a script is also the best option for inserting the content of your use case into your test script (in the form of comments). See the example below, which uses numbered notation format:

   ```
   100.001 In the “Microsoft Word” window, navigate the menu to File, Open….
   100.002 [The “Open” dialog window shows]
   100.003 Select Test.doc.
   100.004 Click Open.
   100.005 [The “Open” dialog window goes away]
   100.006[The “Microsoft Word” window has focus again]
   ```
4. You can now copy/paste these contents into your BDL script as follows:.

   ```
   // 100.001 In “Microsoft Word" window,
       hwndWordMainWindow := CitrixSearchWindow(“*Microsoft
       Word”, MATCH_Wildcard);
       CitrixWindowBringToTop(hwndWordMainWindow);

   // navigate the menu to File, Open….
       CitrixKey(KEY_Alt);
       CitrixKeyString("f");
       CitrixKeyString("o");
   ```

```
// 100.002 [The “Open” dialog window shows]
    hwndOpenDialog := CitrixWaitForWindowCreation(“Open”,
    Match_Exact);

// 100.003 Select Test.doc.
    CitrixMouseClick(150, 100, hwndOpenDialog, MOUSE_
    ButtonLeft);

// 100.004 Click Open
    CitrixMouseClick(300, 200, hwndOpenDialog, MOUSE_
    ButtonLeft);

// 100.005 [The “Open” dialog window goes away]
    CitrixWaitForWindow(hwndOpenDialog, EVENT_Destroy);

// 100.006 [The “Microsoft Word” window has focus again]
    CitrixWaitForWindow(hwndWordMainWindow, EVENT_Activate);
```

If the script fails, these comments will help you determine where the script failed in reference to the use case.

Recording Tips

When recording a terminal services script, you must be aware that unplanned events can change the position and size of target windows during replay, causing test runs to fail. Here are some general tips that will help you avoid the effort of finding and fixing such replay errors.

Try launching the application via the Windows **Run** command. Launching the application via a desktop icon or mouse click may also work, but consider if you will be moving the application to a different server, or if the GUI will vary between users, in which case a desktop icon might appear in a different location. It is therefore recommended that you type the path to the application's EXE file in the Windows **Run** command window.

Keyboard Shortcuts

Mouse clicks require x/y coordinates, a window reference, and the target window must have focus. While GUI testing technologies for tracking mouse clicks have become increasingly reliable, Windows remains a dynamically changing environment, so you must build scripts that can tolerate minor inconsistencies. One way to do this is to use keyboard shortcuts. For shortcuts, the only requirement is window focus.

Note: The **Windows Logo** key is not supported, as it moves the focus away from the Citrix Recorder.

Almost all menu systems in Windows applications have **Alt** key combinations that can be used to initiate specific menu commands. To record use of an **Alt** key shortcut, first give your application focus, then press the **Alt** key, this gives the menu system focus. Notice that many of the menu items have one of their letters underlined. This indicates which **Alt** key commands are available for specific menu items. Press appropriate letter keys to gain access to sub-menus until you reach the menu item that you want to initiate. In some cases, a menu item itself will have a key combination. For example, to open a document in Microsoft Word use the `Ctrl+O` combination. This shortcut does not require that the menu have focus.

Maximizing Windows

If keyboard shortcuts are not available and you must use the mouse, you can reduce the chance of triggering inaccurate mouse clicks by maximizing the window that you are working with. This places the window in a fixed position each time it is used and, in turn, makes mouse clicks more accurate. As a general rule, if a window allows for maximization, it is recommended that you take advantage of it.

Note: `Alt-Space, X` is the Windows keyboard combination for maximizing an active window

When a window does not allow for maximization, during the recording session, move the window to the upper left-hand corner of the desktop before clicking. When recording, the recorder may not be able to pick up on the window handle that you have in the foreground, so having the coordinates relative to the desktop is a must.

Keeping the System Clean

To eliminate unexpected events during terminal services sessions, ensure that you keep your Windows desktop as uncluttered as possible. Imagine, for example, that while your script is replaying, a Windows network message appears. Your script will break because the subsequent mouse click will go to the Windows network message window instead of your intended application window.

Ensure Correct Focus

Before clicking a window, ensure that the window you intend to click exists and has focus. Here is a sample BDL function that helps automate this process:

```
function MyCitrixWaitForWindowCreationAndActivation(sCaption:string;
                nMatch    : number optional;
                nStyle    : number optional;
                nX        : number optional;
                nY        : number optional;
                nWidth    : number optional;
                nHeight   : number optional
                ) : number
var
    hwndNewWindow  : number;  // hwnd of the new window created
    nRTT:number;
begin
    //
    // Check if window exists
    //
        hwndNewWindow := CitrixSearchWindow(sCaption, nMatch);
          hwndNewWindow := 0;
    //
    // If window doesn’t exist, wait for creation
    //
        if (hwndNewWindow < 1) then
            hwndNewWindow := CitrixWaitForWindowCreation(sCaption, nMatch,
            nStyle, nX, nY, nWidth, nHeight);
        end;
    //
    // Check if window is already active, wait if it’s not active
    //
        MyCitrixWaitForWindowActivate(hwndNewWindow);
        MyCitrixWaitForWindowCreationAndActivation:=hwndNewWindow;

end MyCitrixWaitForWindowCreationAndActivation;
```

Using Parameters

If you input the same information more than once, the system under test caches the data in memory rather than repeating the same work. For this reason it is important that you vary input/request data so that the work is as realistic as possible. For example, use different user accounts that have different data ranges. This can be accomplished using a CSV (comma separated values) file as input for a test.

Adding Verifications

Verifications are checks that are added to code to check if the responses that are received from the server are correct. When performing GUI based testing, verifications are automatically added to your script via window synchronizations, however it is recommended that you add additional verifications to your code.

Adding Timers

Custom timers are critically important for tracking an application’s response times. Without custom timers, you cannot determine your end user’s overall experience with an application. Metrics can help determine which aspects of your application are performing well and which are not.

Before you begin adding custom timers to your test script, identify the critical areas of your application and determine where your `MeasureStart` and `MeasureStop` calls should be placed. For this effort it is good practice to review the log in TrueLog Explorer.

Here is an example of using timers in a Silk Performer script:

```
//
// Submit a work order.
//
CitrixMouseClick(27, 31, hwndWorkOrder, MOUSE_ButtonLeft);

//
// Start the response time clock.
//
MeasureStart("202.01: Work Order Submission.");

//
// Wait for the "Order Submission Complete" dialog box.
//
MyCitrixWaitForWindowCreationAndActivation(
    "Order Submission Complete",
    MATCH_Exact
    );

//
// Stop the response time clock.
//
MeasureStop("202.01: Work Order Submission ");
```

Debugging Scripts

Windows may fail to be activated and screen synchronizations may fail when Silk Performer Citrix replay encounters different values during replay than were captured during recording. Sometimes the causes of synchronization problems are not apparent. They may be due to a change in screen position of only a single pixel.

More common than screen synchronization failures are windows not being activated during replay. In such cases the screenshots associated with the corresponding user actions may explain the fault.

Note: Sometimes there is no user fault and a window is activated only sporadically. In such cases you must remove the associated `CitrixWaitForWindow` function.

Silk Performer Citrix replay captures screengrabs when errors occur (the default setting) and writes the bitmaps to disk. By default, the recorder writes screenshots to the screenshots directory in the project directory. Replay stores screenshots in the current used result directory. Visual comparison of record and replay screens is best achieved using bitmap viewing programs.

The Silk Performer **Dump window region of unsuccessful screen synchronizations** Citrix option must be activated (the default) to have bitmaps captured and saved.

Troubleshooting Scripts

You may encounter timeout errors or other execution failures during load test execution. Here are some tips for avoiding, finding, and fixing such errors.

Using TrueLog Explorer

Test runs often fail due to the appearance of unexpected dialogs, causing scripts to lose focus on the windows they are working on. It is therefore recommended that you enable the TrueLog On Error feature in

Silk Performer before you execute tests. Then, if an error occurs, you will be able to visually track it down in TrueLog Explorer.

Clearing Terminal Server Sessions

After a failed test execution, ensure that you reset all terminal server sessions before you execute the test again, otherwise your script will likely fail.

Handling Application Errors

During tests, your application is likely to throw errors due to generated load. Adding event handlers to your script that can handle and report such errors is very helpful.

Here is an example of an event handler that continuously watches for a window with the name **Program Error Intercepted** and executes an `ALT-C` to close any such window that appears. The event handler also generates an error message whenever such an error occurs.

```
dclevent
  handler Handler1 <EVENT_CITRIXINTERRUPT>
  var
    nInterrupt, nWindow : number;
    nStyle              : number;
    sWindowCaption      : string;
    begin
      CitrixGetActInterrupt(nInterrupt, nWindow);
      ErrorAdd(FACILITY_CITRIXENGINE, 47, SEVERITY_INFORMATIONAL);
      print(string(nWindow));
      CitrixGetWindowCaption(nWindow, sWindowCaption);
      if sWindowCaption = "Program Error Intercepted" then
        CitrixKey(67, MOD_Alt); // 'c'
      end;
      ErrorRemove(FACILITY_CITRIXENGINE, 47);
    end Handler1;
```

Avoiding Think Times

While it may be tempting to use `Wait()` or `ThinkTime` statements to avoid having scripts overrun applications under test, this practice is not recommended for two reasons:

- When load generation increases, application processing speed may slow considerably. The wait statement may eventually be too short and the problem of overrunning the application will again present itself.
- If you are measuring the response times of window, text, or screen synchronizations, the time in the `Wait()` statement will artificially bloat response times.

The solution is to use synchronizations.

Re-recording Portions of a Script

Sometimes changes in application behavior result in portions of a recorded script becoming obsolete. Re-recording a portion of a use case is an option, but beware that the recorder may not be able to track the handles of the existing windows, resulting in incorrect window handle numbers. These issues can make the process of integrating newly recorded script with an original script quite tedious. When a use case is small, it is recommended that you re-record the entire use case. Otherwise, follow the process outlined below to integrate a new script with an outdated script:

1. Comment out the section of code that is to be re-recorded.
2. Match the location of the code section requiring replacement with the corresponding section in the use case.
3. Bring a terminal services session up to the corresponding point in the use case.
4. Begin recording the open terminal services session.

5. Perform the actions of the use case that require replacement.
6. Stop the recorder (a script from the recording is then produced).
7. Replace the window handle variables in the newly recorded script with the respective handles of the original script. You may use functions such as `CitrixSearchWindow()` to locate correct window handles.
8. Copy the edited code into the original script.

Handling Citrix Dialog Boxes

Depending on how you connect to Citrix terminal services sessions and your licensing setup, you may see one or both of the following dialog boxes:

- ICA Seamless Host Agent
- Citrix License Warning Notice

These dialog boxes are informational and may or may not appear when you initially log into terminal services sessions. Following are two ways of handling these dialog boxes.

Handling Citrix dialog boxes (Solution #1)

This solution creates an interrupt that handles the dialog boxes if they appear:

```
transaction TMain
  var
  begin
    CitrixInit(800, 600);
    CitrixAddInterrupt(INTERRUPT_WindowCreate, "ICA Seamless Host Agent",
MATCH_Exact);
    CitrixConnect("lab74", "labadmin", "labpass", "testlab1", COLOR_16bit);
    CitrixWaitForLogon();
    hWnd4 := CitrixWaitForWindowCreation("", MATCH_Exact, 0x96840000, -2,
572, 804, 30);
    CitrixWaitForWindowCreation("Program Manager");
    CitrixMouseClick(36, 17, hWnd4, MOUSE_ButtonLeft, MOD_None, -1, 0);
    hWnd11 := CitrixWaitForWindowCreation("", MATCH_Exact, 0x96400000, 2,
313, 163, 263);
    CitrixMouseClick(62, 247, hWnd11, MOUSE_ButtonLeft);
    CitrixWaitForWindow(hWnd11, EVENT_Destroy);
    hWnd12 := CitrixWaitForWindowCreation("Shut Down Windows", MATCH_Exact,
0x94C808CC, 191, 136, 417, 192);
    CitrixWaitForWindow(hWnd12, EVENT_Activate);
    CitrixMouseClick(203, 170, hWnd12, MOUSE_ButtonLeft);
    CitrixWaitForDisconnect();
  end TMain;

dclevent
  handler Handler1 <EVENT_CITRIXINTERRUPT>
  var
    nInterrupt, nWindow : number;
    nStyle              : number;
  begin
    CitrixGetActInterrupt(nInterrupt, nWindow);

    ErrorAdd(FACILITY_CITRIXENGINE, 47, SEVERITY_INFORMATIONAL);
    print(string(nWindow));
    if CitrixGetWindowStyle(nWindow, nStyle) and (nStyle <> 0xB4000000) then
      CitrixWaitForWindow(nWindow, EVENT_Activate);
      CitrixMouseClick(201, 202, nWindow, MOUSE_ButtonLeft);
      CitrixWaitForWindow(nWindow, EVENT_Destroy);
    end;
    ErrorRemove(FACILITY_CITRIXENGINE, 47);

  end Handler1;
```

Handling Citrix dialog boxes (Solution #2)

This sample code loops for 30 seconds, waiting for the Citrix dialog boxes to appear. If the dialog boxes appear, this code closes them.

```
function MyCitrixStartup(nMaxWait: number optional): boolean
    var
        hwndICAHandle              : number;
        hwndFoundLicenseWarning    : number;
        nCount                     : number;
    begin
        hwndICAHandle:=-1;
        hwndFoundLicenseWarning:=-1;
        nCount:=0;

        if (nMaxWait = 0) then
            nMaxWait:=10;
        // if no wait time was passed,
        // use 10 tries (seconds) as a default
        end;

        MeasureStart("MyCitrixStartup");

    //
    // loop until we've handled the conditions or we've tried
    //

        while ((nCount < nMaxWait) and ((hwndICAHandle <=0) or
            (hwndFoundLicenseWarning <=0))) do

    //
    // Just a little feedback, every 10 tries
    //
            if ((nCount MOD 10) =0) then
                print(string(nCount) + ")MyCitrixStartup "
                + " vUser:" + string(GetUserId())
                + " hwndICAHandle=" + string(hwndICAHandle)
                + " hwndFoundLicenseWarning="
                + string(hwndFoundLicenseWarning),
                OPT_DISPLAY_ERRORS , TEXT_GREEN );
            end;
    //
    // if we haven't handled this window yet
    //
            if (hwndICAHandle  <=0)
then
                hwndICAHandle := CitrixWaitForWindowCreation
                ("ICA Seamless Host Agent", MATCH_Exact, 0x94C800C4,
                0, 0, 0, 0, false, 1, true);

                if (hwndICAHandle > 0) then
                    if (CitrixWindowBringToTop(hwndICAHandle)) then
                        CitrixKey(KEY_ENTER); // press ok to close the dialog
                        CitrixWaitForWindow(hwndICAHandle, EVENT_Destroy);
                        // wait for the close
                    end; // end waiting for window to top
                end; // end if we have a valid handle
            end; // if window has not been found yet

            if (hwndFoundLicenseWarning  <=0)
then
                hwndFoundLicenseWarning := CitrixWaitForWindowCreation
                ("Citrix License Warning Notice", MATCH_Exact, 0x94C800C4,
                0, 0, 0, 0, false, 1, true);
                if (hwndFoundLicenseWarning > 0) then
                    if (CitrixWindowBringToTop(hwndFoundLicenseWarning)) then
                        CitrixKey(KEY_ENTER);  // Press ok
                        CitrixWaitForWindow(hwndFoundLicenseWarning,
                            EVENT_Destroy);
                        // wait for it to go away
                    end; // end waiting for window to top
                end; // end if we have a valid handle
            end; // if window has not been found yet

            nCount :=nCount+1;
            Wait 1.0;
        end; // while nCount

        MeasureStop("MyCitrixStartup");
        //
        // return true if we handled any one of these conditions
        //

        MyCitrixStartup2 := (hwndFoundLicenseWarning > 0)
            or (hwndICAHandle > 0) ;

          print("MyCitrixStartup "
            + " vUser:" + string(GetUserId())
            + " Waited " + string(nCount) + " of " + string(nMaxWait)
            + " hwndICAHandle=" + string(hwndICAHandle)
            + " hwndFoundLicenseWarning=" + string(hwndFoundLicenseWarning),
            OPT_DISPLAY_ERRORS , TEXT_GREEN );

end MyCitrixStartup;
```

.NET Support

Silk Performer's .NET support includes the testing of Web Services, .NET Remoting objects, NUnit, and more.

.NET Framework Support

Silk Performer ships with a .NET Framework that allows you to test Web Services and .NET components. This framework includes a set of Benchmark Description Language (BDL) API functions and an Add-On for Visual Studio .NET.

Refer to the *Silk Performer .NET Framework Developer Guide* for detailed information on Silk Performer .NET Framework support.

Using .NET Framework Functions Within .NET

For best results using .NET test code with Silk Performer, call BDL functions from within .NET code. The .NET assembly `perfdotnetfw.dll` that comes with Silk Performer implements the Silk Performer.Bdl class, where most BDL functions are implemented.

1. Add a reference to `perfdotnetfw.dll` in your .NET project.

   This DLL can be found in the Silk Performer installation directory.
2. Import the Silk Performer namespace (see the necessary code below, depending on your implementation language).
   ```
   using Silk Performer; // C#
   imports Silk Performer // VB.NET
   ```
3. Use the static functions in your .NET Code.

For example:

```
Bdl.Print("This is a message from within a .NET Assembly");
```

The functions that are defined by the BDL class can only be used if the .NET assembly runs in a virtual user. This is because perfrun.exe functions (virtual user process) will be called.

Sample Bdf Script

```
dcltrans
  transaction TMain
  var
hObject, hObject2 : number;
hReturn : number;
  begin
    DotNetSetString(hObject, "ConstrValue1");
    hObject := DotNetLoadObject("bin\\Release\\TestDriver.dll",
"TestDriver.TestClass");
    hObject2 := DotNetLoadObject("bin\\Release\\TestDriver.dll",
"TestDriver.ParamClass");
    DotNetSetFloat(hObject, 1.23);
    DotNetSetInt(hObject, 123);
    DotNetCallMethod(hObject,"TestMethod");
    DotNetGetObject(hObject, hReturnValue);
    DotNetFreeObject(hObject);
    DotNetFreeObject(hObject2);
    DotNetFreeObject(hReturn);
  end TMain;
```

Sample .NET Code (C#)

```
using System;
using Silk Performer;
namespace TestDriver
{
  public class TestClass
{

public TestClass(string sConstrValue)
{
Bdl.Print("Constructor called with param" + sConstrValue);
}

public TestClass()
{
Bdl.Print("Default Constructor called!");
}

public ParamClass TestMethod(double fParam, int nParam)
{
return new ParamClass(fParam, nParam);
}
}

  public class ParamClass
{
public double mfMember;
public int mnMember;

public ParamClass(double fParam, int nParam)
{
mfMember = fParam;
mnMember = nParam;
}

public ParamClass()
{
mfMember = 0.0;
mnMember = 0;
}
}
}
```

.NET Configuration Files

Microsoft .NET Framework allows for the specification of configuration files for Windows-, Web- and Internet Explorer-based .NET applications. These configuration files contain settings that affect the .NET runtime environment (e.g., assembly binding and .NET Remoting settings).

When running a .NET test driver with Silk Performer .NET Framework, a configuration file called `perfrun.exe.config` is created automatically with settings that are required for loading the Silk Performer Framework Assembly and allowing the routing of HTTP traffic over the Silk Performer Web Engine.

You have two options for delivering custom configurations through such configuration files:

- Load a configuration file at runtime into your .NET test driver.
- Create a app.config file in your project directory that contains configuration keys that will be merged into the automatically generated `perfrun.exe.config` file.

Configuration at Runtime

The .NET Class Library provides a class that allows you to configure specific .NET Remoting settings that are loaded from configuration files. If for example you have the following configuration file schema:

```
<configuration>
<system.runtime.remoting>
<application>
  <channels>
    <channel ref="http" port="2000" />
  </channels>
  <client url="http://remoteserver:2000">
    <activated type="RemoteDll.RemoteClass1, RemoteDll"/>
    <activated type="RemoteDll.RemoteClass2, RemoteDll"/>
  </client>
</application>
</system.runtime.remoting>
</configuration>
```

…and you store this in a file in your project directory and add it to your data file section you can use the following code to load it at runtime (most likely in the method that corresponds to your init transaction):

```

RemotingConfiguration.Configure(Bdl.GetDataFilePath("vuser.config"));
```

Other settings that can be configured in the application configuration file can be changed through other classes of the .NET Framework, for example, System.AppDomain. For a detailed description of the classes that allow for modification of the .NET Runtime Environment, see the MSDN Online Reference.

Own Configuration File

When running a test using the Silk Performer .NET Framework, Silk Performer automatically generates a configuration file that is used to initialize the .NET runtime that is hosted in the virtual user process. This file is called `perfrun.exe.config` and is stored to the Silk Performer installation directory. The file has a schema like the following:

```
<configuration>
<system.net>
```

```
...
</system.net>
<runtime>
...
</runtime>
</configuration>
```

When generating this configuration file, Silk Performer looks for an `app.config` file in your project directory. If there is such a file, the content of the file is merged just after the runtime node. This means that you can define any required configuration XML elements in your `app.config` file. But you must ensure that the file doesn't have the root `configuration` tag or one of the tags generated by Silk Performer (`system.net` or `runtime`).

Here is a sample configuration file for .NET Remoting components:

```
<system.runtime.remoting>
<application>
<channels>
  <channel ref="http" port="2000" />
</channels>
<client url="http://remoteserver:2000">
  <activated type="RemoteDll.RemoteClass1, RemoteDll"/>
  <activated type="RemoteDll.RemoteClass2, RemoteDll"/>
</client>
</application>
</system.runtime.remoting>
```

Testing Web Services and .NET Remoting Objects with .NET Framework

With the .NET Framework and Silk Performer's Visual Studio .NET Add-On, you can easily test Web services and .NET Remoting objects.

The Framework offers the following options for testing Web services/.NET Remoting objects:

- Writing your test code in Microsoft Visual Studio using a popular .NET Language (C# or VB.NET)
- Making use of Microsoft Visual Studio's Web Service Client Proxy Generation Wizard
- Coding your Web service calls in your .NET test driver
- Running a load test using your .NET test driver
- HTTP traffic that is generated by .NET to call Web service/.NET Remoting methods is redirected over the Silk Performer Web engine, enabling the advantages of Silk Performer's Web engine (modem simulation, IP-address multiplexing, etc.).
- Exploring HTTP traffic (SOAP, XML, and binary) with TrueLog Explorer

Testing Web Services

1. Create a Silk Performer .NET project.

   Create a project either from within Silk Performer (Project type: Web Services/.NET) or create a new .NET project by launching the Silk Performer Project Wizard from Visual Studio .NET (**File** > **New** > **Project**).
2. Create a WebService client proxy.

   Use the Visual Studio .NET Web Service Client Proxy Wizard to generate a client proxy class:

   a) Call Add Web Reference.
   b) Insert the URL to your Web service in the **Address** field.
   c) Click **Add Reference**. A proxy class is then generated with the name of your WebService. Namespace is the name of the Web server where the service is hosted. If your URL is, for example, http://localhost/MyWebService/Service1.asmx then the full name of your proxy class would be localhost.Service1.

3. **Code your WebService calls.**

   Instantiate an object of your proxy class and then make calls to the service in one of your transactions. A good design decision would be to define the proxy as a member variable of your test class, instantiate it in your init transaction and make calls in your main transaction.

   Example (C#):

```
using System;
using Silk Performer;
namespace SPProject1
{
[VirtualUser("VUser")]
public class VUser
{
public localhost.Service1 mService;

public VUser()
{
}

[Transaction(ETransactionType.TRANSTYPE_INIT)]
public void TInit()
{
mService = new localhost.Service1();
}

[Transaction(ETransactionType.TRANSTYPE_MAIN)]
public void TMain()
{
int nRetParam = mService.ServiceCall1("Testparam");
Bdl.Print("Return value of ServiceCall1: " +
nRetParam.ToString());
mService.ServiceCall2(nRetParam);
}

[Transaction(ETransactionType.TRANSTYPE_END)]
public void TEnd()
{
}
}
}
```

4. **Run a TryScript.**

   Initiate a Try Script run by calling Silk Performer/Try Script or by pressing the F8 key.The return value is output into the Virtual User Output Window via the Bdl.Print method.

   In the TrueLog, two nodes in the main transaction represent the SOAP HTTP traffic that was responsible for the Web service calls. By default all HTTP traffic is redirected over the Silk Performer Web Engine; therefore output is available in the TrueLog. You can turn off redirection or enable it for specific Web service client proxy classes using the **Web Settings** dialog.

5. **Continue working in Silk Performer.**

   When you have finished implementing your .NET test driver, you can continue working in Silk Performer and running load tests. With Silk Performer you can run load tests with multiple users distributed over multiple agents. Take advantage of the Web engine features (modem simulation, IP-address multiplexing, etc.) by testing how Web service calls perform when they are called over a slow modem and how your Web server performs when numerous users make service calls simultaneously.

Testing .NET Remoting Objects

1. **Reference remote object assembly.**

   To test .NET Remoting objects you must reference the .NET assembly that defines the objects that are remoted. To do this, add a reference to the DLL that defines the classes.

2. Configure your test driver to be a remoting client.

You must tell .NET which classes are remoted. The easiest way to do this is to make the configuration in a .config file. For detailed information about .config files that define a remoting client configuration, see MSDN.

Here is an example file:

```
<configuration>
 <system.runtime.remoting>
 <application>
 <channels>
 <channel ref="http" port="2000" />
 </channels>
 <client url="http://remoteserver:2000">
 <activated type="RemoteDll.RemoteClass1, RemoteDll" />
 <activated type="RemoteDll.RemoteClass2, RemoteDll" />
 </client>
 </application>
 </system.runtime.remoting>
 </configuration>
```

This configuration file must be loaded at runtime (ideally in the Init transaction of your test driver). To make sure that this file is available on all agents when running load tests, add it to the project-dependent files (Menu: Silk Performer/Add Dependencies). You can get the correct path using the Bdl.GetDataFilePath method.

Here is example code that configures remoting and instantiates a remote object:

```
using System;
using Silk Performer;
using System.Runtime.Remoting;
using RemoteDll;
namespace SPProject1
{
[VirtualUser("VUser")]
public class VUser
{
public VUser()
{
}

[Transaction(ETransactionType.TRANSTYPE_INIT)]
public void TInit()
{
 RemotingConfiguration.Configure(Bdl.GetDataFilePath("vuser.config"));
 }

[Transaction(ETransactionType.TRANSTYPE_MAIN)]
public void TMain()
{ // .NET Runtime knows that objects of type RemoteDll.RemoteClass1 are
remoted
 RemoteClass1 rm1 = new RemoteClass1();
 rm1.SomeMethod("param");
}

[Transaction(ETransactionType.TRANSTYPE_END)]
public void TEnd()
{
}
}
}
```

As HTTP was configured as the transporting protocol, you can view the traffic generated for the .NET remoting calls via TrueLog Explorer.

Testing SOAP Web Services for .NET

A Web service is an available service on the Web that can be invoked and from which results can be returned. Although other standards exist, the widely accepted standard for Web services, which has been adopted by the W3C, is SOAP (Simple Object Access Protocol).

This section explains the basics of SOAP-based Web services and details how you can test them.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol that is used for the exchange of information in decentralized, distributed application environments. You can transmit SOAP messages in any way that the applications require, as long as both the client and the server use the same method. The current specification describes only a single transport protocol binding, which is HTTP.

SOAP perfectly fits into the world of Internet applications and promises to improve Internet inter-operability for application services in the future. In essence, SOAP packages method calls into XML strings and delivers them to component instances through HTTP.

SOAP XML documents are structured around root elements, child elements with values, and other specifications. First an XML document containing a request (a method to be invoked and the parameters) is sent out. The server responds with a corresponding XML document that contains the results.

SOAP is not based on Microsoft technology. It is an open standard drafted by UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft, and SAP. SOAP 1.1 was presented to the W3C in May 2000 as an official Internet standard. Microsoft is one of the greatest advocates of SOAP and has incorporated SOAP as a standard interface in the .NET architecture.

A SOAP stack, an implementation of the SOAP standard on the client side, is comprised of libraries and classes that offer helper functions. A significant Web service testing challenge is that there are a number of SOAP stack implementations that are not compatible with one another. So although SOAP is intended to be both platform- and technology-independent, it is not. Web services written in .NET are however always compatible with .NET clients—they use the same SOAP stack, or library. When testing a .NET Web service however, you need to confirm if the service is compatible with other SOAP stack implementations, for example Java SOAP stack, to avoid interoperability issues.

SOAP client requests are encapsulated within HTTP POST or M-POST packages. The following example is taken from the Internet draft-specification.

Sample Call

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The first four lines of code are standard HTTP. POST is the HTTP verb which is required for all HTTP messages. The `Content-Type` and `Content-Length` fields are required for all HTTP messages that contain payloads. The content-type `text/xml` indicates that the payload is an XML message to the server or a firewall capable of scanning application headers.

The additional HTTP header `SOAPAction` is mandatory for HTTP based SOAP messages, and you can use it to indicate the intent of a SOAP HTTP request. The value is a URI that identifies the intent. The content of a `SOAPAction` header field can be used by servers, for example firewalls, to appropriately filter SOAP request messages in HTTP. An empty string ("") as the header-field value indicates that the intent of the SOAP message is provided by the HTTP Request-URI. No value means that there is no indication on the intent of the message.

The XML code is straightforward. The elements `Envelope` and `Body` offer a generic payload-packaging mechanism. The element `GetLastTradePrice` contains an element called `symbol`, which contains a stock-ticker symbol. The purpose of this request is to get the last trading price of a specific stock, in this case Disney (DIS).

The program that sends this message only needs to understand how to frame a request in a SOAP-complient XML message and how to send it through HTTP. In the following example, the program knows how to format a request for a stock price. The HTTP server that receives the message knows that it is a SOAP message because it recognizes the HTTP header `SOAPAction`. The server then processes the message.

SOAP defines two types of messages, *calls* and *responses*, to allow clients to request remote procedures and to allow servers to respond to such a request. The previous example is an example of a call. The following example comes as a response in answer to the call.

Sample Response

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The first three lines of code are standard HTTP. The first line indicates a response code to the previous POST request, the second and third line indicate the content type and the fourth line the lenght of the response.

XML headers enclose the actual SOAP payloads. The XML element `GetLastTradePriceResponse` contains a response to the request for a trading price. The child element is `Price`, which indicates the value that is returned to the request.

Testing SOAP Over HTTP-Based Web Services

Silk Performer offers three options for testing SOAP over HTTP based services:

- Recording/replaying HTTP traffic
- .NET Explorer in combination with Silk Performer .Net Framework
- Java Explorer in combination with Silk Performer Java Framework

Your environment and prerequisites will determine which of these options is best for your needs.

Recording and Replaying HTTP Traffic

Recording the SOAP protocol over HTTP is as straightforward as recording any Web application that runs in a browser. The application that you record is the application that executes the SOAP Web Service calls. This can either be a client application or a part of the Web application itself.

Creating a New XML/SOAP Project

When you want to record and replay HTTP traffic to test SOAP over HTTP-based Web services, you first need to create a new Silk Performer project of the **Web Services** > **XML/SOAP** type.

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 Services** > **XML/SOAP**. This application type automatically configures its profile settings so that `SOAPAction` HTTP-headers, that are used by SOAP-based applications when calling Web services, are to be recovered.
5. Click **Next** to create a project based on your settings.

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

Creating the Recording Profile

When you want to record and replay HTTP traffic to test SOAP over HTTP-based Web services, you need to create a recording profile for the client application that you want to record.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Click **Settings**. The **System Settings - Recorder** dialog box appears.
3. In the **System** group box, click **Recorder**. The **Recording Profiles** page opens.
4. Click **Add** to add a new recording profile to the list. The **Recording Profile** dialog box opens.
5. Type a name for the recording profile in the **Profile name** text box.

   For example `Internet Explorer`.
6. Click **Browse ...** next to the **Application** path text box and select the path to the application executable.

   For example `C:\Program Files\Internet Explorer\Explorer.exe`.
7. Define the **Working directory**.

   For example `C:\Program Files\Internet Explorer`.
8. Define the **Program arguments**.

   For example `about:blank`.
9. Select the application type from the **Application Type** list box.

   For example `MS Internet Explorer`.
10. In the Protocol selection area, check the check box that corresponds to the protocol that you want to use. For example, check the **Web** check box.
11. To configure the recording profile for WinSock recording click **Web Settings**, which enables you to select the method that the Recorder is to use to capture Web and TCP/IP-based traffic.
12. Click **OK**.

Recording a Script

Record a script with your created recording profile. Interact with your client application and the recorder will record all SOAP requests that are executed over HTTP/HTTPS. When you are finished, close the application and save the recorded script.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select one of the listed browsers from the **Recording Profile** list, depending on the browser you want to use for recording.
3. In the **URL** field, enter the URL that is to be recorded.
4. Click **Start recording**. The Silk Performer Recorder dialog opens in minimized form, and the client application starts.
5. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.
6. Interact with your client application. The recorder records all SOAP requests that are executed over HTTP/HTTPS.
7. To end recording, click the **Stop Recording** button.
8. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.

Script Customization

Each SOAP request that is recorded includes a `WebHeaderAdd` and a `WebUrlPostBin` API call.

You can either customize the input parameter of each Web Service call by manually changing the script or you can use the more convenient method of performing customizations within TrueLog Explorer. To do this, run a Try Script. Then use the XML control to customize the XML nodes that represent the input parameters.

Sample SOAP Request

```
WebHeaderAdd("SOAPAction", "\"http://tempuri.org/Login\"");
WebUrlPostBin(
  "http://localhost/MessageWebService/MessageService.asmx",
  "<?xml version=\"1.0\" encoding=\"utf-8\"?>"
  "<soap:Envelope xmlns:soap=\"http://schemas.xmlsoap.org/soap/
envelope/
\""
    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\""
    "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">"
    "<soap:Body>"
      "<Login xmlns=\"http://tempuri.org/\">"
        "<sUsername>myuser</sUsername>"
        "<sPassword>mypass</sPassword>"
      "</Login>"
    "</soap:Body>"
  "</soap:Envelope>", STRING_COMPLETE, "text/xml;
charset=utf-8");
```

Replaying a Script

Once you have finished script customization, you can replay your script, either in another Try Script run, as part of baseline identification, or in a test.

Select how you want to replay your script. The following options are available:

- Start a Try Script run.
- Replay the script as part of a baseline identification.

- Replay the script in a test.

  As the Web service calls are performed along with Web API functions, you receive the same measures you receive when testing any Web application, including detailed protocol-specific statistics.

Silk Performer .NET Explorer

This section offers a brief overview of how you can use .NET Explorer to test Web services with Silk Performer. .NET Explorer is a tool that allows you to create test cases through a point and click interface. .NET Explorer provides support for the following .NET technologies:

- SOAP Web services
- .NET Remoting
- .NET Components (other classes)

You can use the .NET Explorer to create test scenarios. You can then use the test scenarios to run component testing in .NET Explorer or you can export the test scenarios to Silk Performer for testing.

You can also use .NET Explorer to generate test drivers, in either C# or VB.NET, that contain all test logic defined for test scenarios. You can export the test drivers to Visual Studio .NET where you can make further customizations and where you can execute TryScript runs directly in Visual Studio .NET. Once you have completed the customization, you can execute tests with Silk Performer.

.NET Explorer requires permanent projects. This was not the case with previous versions as it was possible to save current projects at any time. When you launch .NET Explorer you are prompted to either create a new project or open an existing project by choosing from a recent file list or by browsing for a .NET Explorer project file, of type `.NEF`.

For additional details on .NET Explorer, refer to the .NET Explorer Help.

Requirements

The only requirement for testing a SOAP-based Web service is a description of the exposed methods of the Web service. You can find such descriptions in Web Service Description Language (WSDL) files, which are normally generated when you browse a Web-service end point or you specify special `HTTP GET` parameters for retrieving such files.

In the case of an ASP.NET Web service, appending `?WSDL` to the URL end point of the Web service will return the WSDL file for the Web service. Other SOAP-stack implementations may use the same approach or offer WSDL files under separate URLs.

Loading a WSDL File

.NET Explorer offers the **Load File Wizard** that guides you through the steps required to load a WSDL file and invoke Web-service methods. To activate the wizard in .NET Explorer, click **Start Here** on the toolbar.

You can also load WSDL files, or .NET assemblies that contain .NET Remoting or other .NET classes, through the address bar of your browser. Specify the URL or path to the WSDL file and click **Load**.

Microsoft .NET Framework offers classes that load WSDL files and generate client proxies for Web services that are defined in WSDL files. .NET Explorer uses this functionality to generate C# or VB.NET code for Web-service proxies and compile the code into temporary .NET assemblies that are displayed as **Web Service Proxies** in **.NET Explorer** > **Loaded Components** > **References**.

As .NET Explorer uses Microsoft .NET Framework to generate proxies, .NET Explorer shares the drawbacks and limitations of Microsoft .NET Framework. The most significant problem when generating proxies is that not all SOAP stack implementations produced by other vendors comply with the W3C standard. This can lead to problems when you attempt to load WSDL files. You can avoid these problems by manually editing the WSDL files so that they are recognized by Microsoft .NET Framework. To edit the WSDL files you must be familiar to WSDL. For additional information, refer to *http://www.w3.org*.

.NET Explorer shows a Web-service proxy class, derived from `System.Web.Services.Protocols.SoapHttpClientProtocol`, in the **References** menu tree below the **Web Service Proxies** tree node. Normally, when you write C# or VB.NET code, you must instantiate an instance of the proxy class and call methods on the proxy. .NET Explorer eliminates the need for this by automatically instantiating an instance of the proxy class. You cannot create an instance of a proxy class by calling the constructor. .NET Explorer treats Web services like static objects offering static methods.

If the methods of a Web service take complex objects as parameters, then the classes of those parameters are defined in the WSDL file and loaded by .NET Explorer. Such classes are not Web-service proxy classes. They are simple classes with members and are listed under the **Other Classes** tree node in the **Class** menu tree.

You can set several connection-related properties by double-clicking a proxy class in the menu tree and opening the connection wizard in the **Input Data Properties** pane. These properties are set to the corresponding properties of the *internal* proxy instance.

When you export a project to either Visual Studio .NET or directly to Silk Performer as a .NET project, the base class of the Web-service proxy is replaced by `SilkPerformer.SPSoapHttpClientProtocol`. The reason for this is that by exchanging the base class, the Silk Performer .NET Framework is able to generate more detailed timers for Web-service calls that are routed through the Silk Performer Web engine. If you don't want this behavior you can export a project to Visual Studio .NET and either change the base class back manually or use the **Web Service** dialog box from the Silk Performer menu and deselect the option for routing the proxy class.

You can now either load the WSDL file of the Web service you want to test or select a WSDL file from the list box.

Calling a Web Service

After you have successfully loaded a WSDL file you can explore the methods that the Web service offers by expanding the tree node of the Web-service proxy class.

By clicking one of the methods you will see the required input parameters and input header information for the Web-service call. You can customize your input data by either exchanging the default static values for the primitive types or by using global, local, or rendom variables. For additional information, refer to the *.NET Explorer Help* and the *Silk Performer Help*.

If a method is called for the first time on a Web service, the internal instance of the proxy class is instantiated. There is a property on the proxy class that holds a cookie container. This property is initialized with a new cookie container so that it can call Web services that handle cookies.

.NET Explorer then sets all the defined values for the SOAP headers to the corresponding member fields of the proxy class. Then a parameter list with all the values that are defined for the input parameters is created. Using this list, the method on the Web-service proxy object is invoked.

Microsoft .NET Framework includes a hooking mechanism that allows .NET Explorer to capture traffic that is passed between the .NET Explorer client and the Web server. You can view the trafic in the **Show Traffic** dialog box after the method call. You can invoke the dialog box on each Web service and each .NET Remoting call.

This feature is also used to generate BDL Web scripts with a `WebPagePost` for each Web-service call captured in the traffic moving from the client to the server.

The returned values and SOAP header information are displayed when the method calls return successfully. When exceptions occur, the exception text is displayed in a message box. Currently you cannot add method calls to test scenarios that throw errors because .NET Explorer requires information about the returned values.

Final Steps

After calling a Web service you can store the returned values to variables and define verifications for those values.

Once you have finished defining your test scenario you can either remain in .NET Explorer, and use your test scenario for functional testing, or you can export the project to Visual Studio .NET or Silk Performer to further customize the generated code and run regression tests.

You can only export to BDL Web projects when your test scenarios contain only Web-service calls, because only `WebPagePost` statements are generated for each Web-service call. If you have calls other than Web-service calls, for example calls to other .NET objects, those calls will not be included in Web scripts and therefore your exported scripts may not behave as defined in .NET Explorer, as some method calls will be missing.

Only export to BDL Web projects if you have only Web-service calls and if you only wish to test the SOAP stack of your server, as there is no .NET client SOAP stack involved when executing scripts in Silk Performer. The Silk Performer Web engine posts only those SOAP envelopes that have been used in .NET Explorer.

If you also wish to test the .NET client side, you should export your project to a Silk Performer .NET project. This type of project will compile generated .NET test code into a .NET assembly that can be called from a BDL script, which will also be generated by .NET Explorer.

If you wish to make further customizations to .NET code generated by .NET Explorer you can export your project to Visual Studio .NET. If you export your project you can alter generated test code and run a TryScript within Visual Studio .NET. If you are finished with customizations you can export the project to Silk Performer and proceed with testing.

Silk Performer .NET Framework

Silk Performer's .NET Framework enables developers and QA personnel to coordinate their development and testing efforts while allowing them to work entirely within their specialized environments: Developers work exclusively in Visual Studio while QA staff work exclusively in Silk Performer—there is no need for staff to learn new tools. Silk Performer's .NET Framework thereby encourages efficiency and tighter integration between QA and development. The Silk Performer .NET Framework (.NET Framework) and .NET Add-On enable you to easily access Web services from within .NET. Microsoft Visual Studio offers wizards that allow you to specify the URLs of Web services. Microsoft Visual Studio can also create Web-service client proxies to invoke Web-service methods.

Creating a New Silk Performer .NET Project

To create a new Silk Performer .NET project, you can either create a new project of type **Web Services** > **.NET Explorer** in Silk Performer or you can use one of the Silk Performer .NET project wizards in Visual Studio .NET.

With both approaches you can choose one of the following three implementation languages:

- C#
- VB.NET
- C++

If you choose one of the .NET project wizards in Visual Studio .NET, the result is a new project with a template class that defines three methods, which are the init, main, and end transactions of your Silk Performer virtual user.

To develop a .NET test driver in another language, create an empty project using the language of your choice and perform the following steps:

1. Add a reference to `perfdotnetfw.dll`.

   This DLL is located in the Silk Performer installation directory.
2. Add a new class to your project.
3. Add the `VirtualUser` custom attribute to your class.
4. Add public member functions to your class to serve as your user transactions.

5. Add the Transaction attribute to the functions you have created and pass the corresponding transaction type.
    - init
    - main
    - end

Creating a Web Service Client Proxy

Visual Studio .NET has a wizard that generates a Web-service-client proxy that allows you to call Web-service methods.

You can start the wizard in **Project** > **Add Web Reference**.

1. To start the wizard, click **Project** > **Add Web Reference**.
2. In the corresponding text box, type the URL of your Web service and press **Enter**.

   For example, *http://demo.borland.com/BorlandSampleService/BorlandSampleService.asmx?WSDL*.
3. If the wizard can load the WSDL document from the URL, click **Add Reference**. The wizard generates a proxy class in a namespace, which is the reverse of the name of the Web server that hosts the service.

Explore projects to see which classes are generated. Each web service, and all complex data types used by the Web-service methods, are represented as classes. So in the example URL above, there is `Service1`, which is a Web service, and `User`, which is a complex parameter.

Instantiating a Client Proxy Object

You can declare a variable of a client proxy class as a public member of the .NET test driver to instantiate a client-proxy object. The variable should be instantiated either in the constructor or in the `init` transaction. The first part of the namespace where the class is generated, which is the default namespace, is the name of your project.

Example

If you have created a project with the name *DotNetProject* you would use the following variable declaration:

```
[VirtualUser("Vuser")]
public class Vuser
{
  public DotNetProject.com.borland.demo.Service1 mService;
  [Transaction(Etranstype.TRANSTYPE_INIT)]
  public void TInit()
  {
    mService = new DotNetProject.com.borland.demo.Service1();
  }
}
```

Calling a Web Service Method

All methods that are exposed by Web services are also available in proxy objects. The methods that are shared by proxy objects use the same names as their corresponding WSDLs. Web-service method calls should be placed in `main` transactions.

Example

```
[Transaction(Etranstype.TRANSTYPE_MAIN)]
public void TMain()
{
  string sReturn = mService.echoString("Test");
  Bdl.Print(sReturn);
}
```

To customize your Web-service calls from a generated BDL script, you must allow the exchange of data between BDL and .NET with usage of attributes or method parameters.

Example

```
[Transaction(Etranstype.TRANSTYPE_MAIN)]
[TestAttribute("EchoInput", "Test")]
public void TMain()
{
  string sReturn =
mService.echoString(Bdl.AttributeGet("EchoInput"));
  Bdl.Print(sReturn);
}
```

or

```
[Transaction(Etranstype.TRANSTYPE_MAIN)]
public void TMain(string sEcho)
{
  string sReturn = mService.echoString(sEcho);
  Bdl.Print(sReturn);
}
```

Routing Web-Service Traffic

The Silk Performer .NET Framework can route Web traffic generated by .NET components through the Silk Performer Web engine. This means that the Web engine executes the actual Web requests, enabling you to see exactly what is sent over the wire. You can also use features of the Silk Performer Web engine, like modem simulation, IP multiplexing, network statistics, TrueLog, and others.

By default all network traffic is routed through the Web engine. You can switch the routing off and only enable it for specific Web-service client-proxy classes. To switch the routing on for specific Web-service client-proxy classes, you need to change the base class of the proxy classes from `SoapHttpClientProtocol` to `SilkPerformer.SPSoapHttpClientProtocol`. Changing the base class allows the Silk Performer .NET Framework to generate more detailed statistical information for each Web-service call. We recommend that you enable this feature for all your Web-service proxy classes. You can enable this feature by using the **Web Service** dialog box in Microsoft Visual Studio, which is accessible through the Silk Performer menu.

For each Web-service call a node is created in the TrueLog with the SOAP envelope that was passed to the Web service and returned to the client.

If detailed statistical information for Web-service calls is disabled, the .NET HTTP classes process all requests.

Exploring Results

When Web-service-traffic routing is enabled, a TrueLog node is logged for each Web-service call that is executed by the .NET test driver.

In the overview report of the Web-service method that is called, you will find statistical information.

External References

1. Session, Roger

SOAP. An overview of the Simple Object Access Protocol, March 2000

http://www.w3.org/TR/SOAP/

2. W3C

   Simple Object Access Protocol (SOAP) 1.1, December 2000

   http://www.w3.org/TR/SOAP/
3. UN/CEFANT, OASIS

   Enabling Electronic Business with ebXML, December 2000

   http://www.ebxml.org/white_papers/whitepaper.htm
4. Geyer, Carol

   ebXML Integrates SOAP Into Messaging Services Specification, March 2001

   http://www.ebxml.org/white_papers/whitepaper.htm
5. Open Financial Exchange

   Open Financial Exchange Specification 2.0, April 2000

   http://www.ofx.net/

Setting Up Silk Performer .NET Projects

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 **.NET** > **.NET Framework using Visual Studio .NET Add-On** and click **Next**. The **Workflow - Model Script** dialog box opens.
5. Select the .NET Language (**C#** or **VB.NET**) icon for the language you prefer and click **OK**. The Microsoft Visual Studio **Silk Performer Project Wizard** opens.
6. Enter the name of the .NET Testclass in the **Name of testclass** text box. In the **Silk Performer Project** text box, enter the name of the project that you created earlier in Silk Performer.
7. Click **Finish**.

The following in the files and code are generated in Microsoft Visual Studio:

- Each generated Testclass becomes a VirtualUser in the BDL script.
- The first transaction becomes the `Init` transaction in the BDL script.
- Files that are generated by the Wizard (code files and Silk Performer project/BDL scripts) are listed on the **Solution Explorer** page.
- Handler/clean-up code can be inserted in the `stopException` method.
- Custom code for exception handling can be inserted in the `testException` method.
- `ETransactionType.TRANSTYPE_MAIN` becomes the `Main` transaction in the BDL script.
- `ETransactionType.TRANSTYPE_END` becomes the End transaction in the BDL script.

Sample Skeleton Code Generated by the Project Wizard (C#)

```
using System;
using Silk Performer;

namespace SPProject1
{
  [VirtualUser("VUser")]
  public class VUser
  {
    public VUser()
    {
    }

    [Transaction(ETransactionType.TRANSTYPE_INIT)]
    public void TInit()
    {
      /* You can add multiple TestAttribute attributes to each
function defining parameters that can be accessed through
Bdl.AttributeGet


      Example of testcode: (Access bdl function through the
static functions of the Bdl class Bdl.MeasureStart(...);
      ...
      Bdl.MeasureStop(...);
      */
    }

    [Transaction(ETransactionType.TRANSTYPE_MAIN)]
    public void TMain()
    {
    }

    [Transaction(ETransactionType.TRANSTYPE_END)]
    public void TEnd()
    {
    }
  }
}
```

As you can see from the skeleton example above, there is a custom attribute called `VirtualUser` that can be applied to classes. This causes the Add-On's BDL Generation Engine to generate a virtual user definition. You can implement multiple classes that have the `VirtualUser` attribute applied. The `VirtualUser` attribute takes the name virtual user as a parameter.

The BDL Generation Engine then parses the methods of the Virtual User class for methods that have a `Transaction` attribute applied to them. The `Transaction` attribute takes as a first parameter the transaction type (`Init`, `Main` or `End`). You can only have one `Init` and one `End` transaction, but multiple `Main` transactions.

The `Main` transaction type takes a second parameter that indicates the number of times that the transaction is to be called during load tests (default: 1).

Running a Try Script Test

If you have implemented your .NET test code you can run a Try Script test from within Microsoft Visual Studio .NET.

1. Choose one of the following:
   - Press F8.
   - Choose **Silk Performer** > **TryScript**.

   The .NET Code is compiled to a .NET assembly.

   A BDF script is generated by the BDL Generation Machine based on the meta information of the custom attributes and the settings in the Options dialog.

   The most recent BDF script is overwritten if there have been changes to the meta data of your assembly (changed custom attributes, changed method order, changed generation options, etc.).

   If the meta data has changed but you have altered the latest BDF file manually, you will be prompted to specify whether or not you wish to have the file overwritten, in which case you will lose recent changes.

This detection is achieved by comparing the last modified date of the BDF file with the timestamp scripted in the BDF file.

2. If you have multiple virtual user classes (classes that have the VirtualUser attribute applied) you will be prompted to specify which of the users is to be started. The test begins.

   If you have the **Automatic Start when running a Try Script** option selected, TrueLog Explorer will start, loaded with the TrueLog generated by the test.

   Virtual user output is displayed in the Virtual User Output tool window.

   Load test controller output is displayed in a separate pane of the Output tool window.

Once the load test is complete you can explore the other results files (log, output, report, and error) by selecting them from the Silk Performer **Results** menu.

Custom Attributes

A custom attribute called `VirtualUser` can be applied to classes. This attribute instructs the Add-In's BDL generation engine to generate a virtual user definition. You can implement multiple classes that have the `VirtualUser` attribute applied to them. The `VirtualUser` attribute takes the name `virtual user` as a parameter.

Note: When a BDF file is modified manually, you are prompted to specify whether or not you want to have the file overwritten.

The BDL generation engine parses the methods of the `VirtualUser` class for methods that have a `Transaction` attribute applied to them. The `Transaction` attribute takes the transaction type, `Init`, `Main` or `End`, as a first parameter. You can only have one `Init` and one `End` transaction, but multiple `Main` transactions are allowed.

The `Main` transaction type takes a second parameter that indicates the number of times that the transaction is to be called during a test (the default is `1`).

Following are the available custom attributes and what the BDL generation engine scripts for them.

Attribute Class	Applicable to	Parameters	Description
VirtualUser	Class	Name of the Virtual User Group (optional) IsUnitTest	Defines a Virtual User Group. If you specify true, DotNetUnitTest methods will be scripted instead of the standard DotNet methods (e.g., DotNetUnitTestLoadObject).
Transaction	Method	Type (Init, Main, End) If type is Main the number of transaction iterations (optional) Name	Defines a Transaction for the Virtual User Group. The transaction implementation will call the method of the .NET Object. The first script call in the Init transaction is a DotNetLoadObject loading the Object The last script call in the end transaction is a DotNetFreeObject. Optionally you can define a name that should be used in the generated BDL script for this transaction. By default, the transaction name in BDL is created by combining the VUser name and the method name.
TestMethod	Method		This will script a call to the method in the current transaction. The current transaction is the previous method with a Transaction attribute. So a method with this attribute that has no prior method with a Transaction attribute makes no sense.

Attribute Class	Applicable to	Parameters	Description
TestAttribute	Method	Attribute Name Attribute Value (optional) Description	This can be applied multiple times to a method that has either a Transaction or TestMethod attribute. An AttributeSetString function will be scripted prior to the DotNetCallMethod that calls this method. AttributeSetString will set an attribute with the passed name and value. This is a way how parameters can be passed from the script to the .NET function. The .NET function can read the attributes with Bdl.AttributeGet. Its meant that people (QA) who will receive the finished script only have to change the value passed to the AttributeSetString to customize the script. So there is no need for them to change the .NET Code. Allows you to define a description for the project attribute. The description can be seen in Silk Performer's project attribute wizard.
VirtualUserInitialize	Method		This method is called for classes that are loaded via DotNetUnitTestLoadObject
VirtualUserCleanup	Method		This method is called for classes that are freed via DotNetUnitTestFreeObject
TestCleanup	Method		This method is called after a method is called via DotNetUnitTestCallMethod
TestInitialize	Method		This method is called before a method is called via DotNetUnitTestCallMethod
TestIgnore	Method		Methods that have this attribute applied to them will not be called via DotNetUnitTestCallMethod
TestException	Method	Type of exception Additional log message	Normally, methods that throw exceptions are considered failed. If you want a method to throw an exception, you can use the TestException attribute to tell Silk Performer that this method is supposed to throw an exception.

Custom Attributes Code Sample

C# Test Code Sample

```
using System;
using SilkPerformer;

namespace SPProject1
{
  [VirtualUser("VUser")]
  public class VUser
  {
    public VUser()
{
    }
    [Transaction(ETransactionType.TRANSTYPE_INIT)]
    public void TInit()
    {
    }
    [Transaction(ETransactionType.TRANSTYPE_MAIN)]
    public void TMain()
    {
    }
    [TestMethod]
    [TestAttribute("Attr1", "DefaultValue1")]
    public void TestMethod1()
    {
      string sAttrValue = Bdl.AttributeGet("Attr1");
      Bdl.Print(sAttrValue);
    }
    [Transaction(ETransactionType.TRANSTYPE_END)]
    public void TEnd()
    {
    }
  }
}
```

Generated BDF Script Example

```
benchmark DOTNETBenchmarkName

use "dotnetapi.bdh"

dcluser
 user

    VUser
 transactions
   VUser_TInit : begin;
   VUser_TMain : 1;
   VUser_TEnd  : end;
var
 hVUser : number;

dcltrans
 transaction VUser_TInit
 begin
   hVUser:= DotNetLoadObject("\\SPProject1\\bin\\release\
\SPProject1.dll", "SPProject1.VUser");
   MeasureStart("TInit");
   DotNetCallMethod(hVUser, "TInit");
   MeasureStop("TInit");
 end VUser_TInit;

 transaction VUser_TMain
 begin
   MeasureStart("TMain");
   DotNetCallMethod(hVUser, "TMain");
   MeasureStop("TMain");
   AttributeSetString("Attr1", "DefaultValue1");
   MeasureStart("TestMethod1");
   DotNetCallMethod(hVUser, "TestMethod1");
   MeasureStop("TestMethod1");
 end VUser_TMain;

 transaction VUser_TEnd
 begin
   MeasureStart("TEnd");
   DotNetCallMethod(hVUser, "TEnd");
   MeasureStop("TEnd");
```

```
  DotNetFreeObject(hVUser);
end VUser_TEnd;
```

Creating Silk Performer .NET Projects in Microsoft Visual Studio

When you create a Silk Performer .NET project from within Microsoft Visual Studio, ensure that the created solution file (`.sln`) is placed in the same directory as the created project file (`.csproj`, `.vbproj`, etc). In Microsoft Visual Studio's **Create Project** wizard, there is a **Create directory for Solution** check box. Do not select this option (i.e., leave it unchecked). Otherwise you may experience problems when opening the project from Silk Performer's **Model Script** dialog.

Memory Usage in .NET

When you run BDL scripts with .NET method calls, Silk Performer needs to host the Microsoft .NET Common Language Runtime (.NET CLR). Hosting the .NET CLR and loading .NET objects requires additional memory that is used for virtual user processes (containers).

Silk Performer allows you to specify whether each virtual user should run in its own application domain or if all virtual users in a virtual user container should share an application domain. One application domain per user means that all objects from individual virtual users are isolated from the objects of the other users in the same virtual user container.

One application domain per virtual user container means that all objects exist in the same domain and may influence one another. This setting can be changed via your project's profile settings.

Each additional application domain requires additional memory and administrative overhead by the .NET CLR.

Memory consumption (in MB) is listed below, based on the number of users running in a virtual user process and the application domain setting. This is initial memory consumption when starting a load test (for example, no .NET objects have been loaded).

	1 VUser	2 VUser	3 VUser	4 VUser	10 VUser	20 VUser	50 VUser
No .NET Calls (normal BDL Script)	1.668	1.760	1.836	1.904	2.352	3.124	5.484
App Domain/ User	5.268	6.644	6.828	6.940	7.916	9.364	13.868
App Domain/ Container	5.268	5.404	5.488	5.564	6.036	6.852	9.368

The next table shows memory consumption after loading a simple .NET object. The object defines three methods, but no members.

	1 VUser	2 VUser	3 VUser	4 VUser	10 VUser	20 VUser	50 VUser
App Domain/ User	5.680	7.404	7.816	8.296	11.000	15.300	28.664
App Domain/ Container	5.688	5.812	5.900	6.012	6.620	7.628	10.240

As you can see from the tables above, having one application domain per virtual user requires more memory than does having one application domain per container. The default setting in the project file is one application domain per user. The reason for this is that the objects created by a user cannot influence the objects of other users. Ensure that you won't run into problems if objects access global resources from within their application domain.

The initial amount of memory needed to host the .NET CLR is about 3.5 MB. The .NET CLR is hosted only once per virtual user container -- this is not influenced by application domain setting.

Memory consumption when loading objects depends on the size of the objects. In the table above, the example loaded object has no members. If you are loading objects with members, memory consumption will be different.

To free up memory from objects that are no longer referenced, call DotNetFreeObject.

.NET Error Messages

The following issues may occur during .NET load tests.

Cannot Start .NET Language Runtime

Possible Cause	Resolution
Silk Performer couldn't host the .NET Common Language Runtime.	Make sure you have installed the .NET Common Language Runtime.

Invalid Handle

Possible Cause	Resolution
You passed an invalid .NET object handle to the DotNet API function.	Object handles can be acquired by a call to `DotNetLoadObject` or they can be received with `DotNetGetObject` if the last called method returns an object.

Loading Object Throws an Exception

Possible Cause	Resolution
This error occurs if: • Referenced assemblies cannot be loaded/found • Your `perfdotnetfw.dll` has a different version than the `perfrun.exe`	Ensure that the assembly files are in the location being referenced and that the versions match for `perfdotnetfw.dll` and `perfrun.exe`.

Object Does Not Implement This Public Method

Possible Cause	Resolution
You attempted to call a method that is not a public method of the .NET object.	Only public methods can be called with the `DotNetCallMethod` function.

Method Execution Failed

Possible Cause	Resolution
Method execution failed because of an unhandled exception.	Examine the method that failed.

.NET Common Language Runtime Not Started

Possible Cause	Resolution
This error occurs when you attempt to call a DotNet function before the .NET CLR has been started.	Ensure that the .NET CLR starts before you call a DotNet function.

Cannot Load mscoree.dll (.NET Runtime)

Possible Cause	Resolution
Silk Performer couldn't host the .NET Common Language Runtime.	Make sure you have installed the .NET Common Language Runtime. The runtime can be installed from the *Microsoft Download Center*.

Initialize .NET Application Domain Failed

Possible Cause	Resolution
A .NET application domain couldn't be started.	Check to see if your user has the privilege to run .NET applications.

.NET Common Language Runtime Is Not Running

Possible Cause	Resolution
This error occurs when you attempt to call a DotNet function before the .NET CLR has been started.	Ensure that the .NET CLR starts before you call a DotNet function.

.NET Assembly With the Given Name Could Not Be Found

Possible Cause	Resolution
The assembly file passed as the first parameter to `DotNetLoadObject` cannot be found.	If you passed a relative path, the path should be relative to the project directory.

Perfrun.Exe.Config Could Not Be Written Successfully

Possible Cause	Resolution
Before Silk Performer hosts the .NET CLR the file `perfrun.exe.config` will be written to set configuration settings for the loaded .NET objects.	Confirm that this file is write-enabled and that the executing user has appropriate privileges.

Your .NET DLL References Wrong perfdotnetfw.dll

Possible Cause	Resolution
Your `perfdotnetfw.dll` has a different version than the `perfrun.exe`	Ensure that the version of your `perdotnetfw.dll` is the same as the `perfrun.exe`.

Verification Failed

Possible Cause	Resolution
This error is raised when a verification with one of the verification functions provided by `perfdotnetfw.dll` fails.	Examine the verification function that failed.

Exception Has Been Logged

Possible Cause	Resolution
This error is raised when you log an exception with the `Bdl.LogException` method provided by `perfdotnetfw.dll`.	Examine the exception that was logged.

Could Not Find or Load a Specific Class

Possible Cause	Resolution
The class with the name you passed as the second parameter to `DotNetLoadObject` cannot be found in the assembly.	Check the spelling and ensure that you defined the full name (including namespace).

An Exception Was Thrown by a Type's Initializer (.cctor)

Possible Cause	Resolution
Constructor of the object has thrown an exception.	Examine the object constructor that threw the exception.

An Exception Was Thrown by the Called Method

Possible Cause	Resolution
The method called with `DotNetCallMethod` has thrown an exception.	Examine the method that threw the exception.

Invalid Number of Parameters

Possible Cause	Resolution
You passed an invalid number of parameters to the `DotNetCallMethod` function.	Check the parameters used by the `DotNetCallMethod` function.

NUnit Integration

Silk Performer facilitates smooth integration of existing NUnit and Microsoft Unit test scripts into Silk Performer for the support of remote-component testing under realistic concurrent-access server conditions.

Silk Performer provides an import tool that enables you to import existing NUnit assemblies and other .NET assemblies. Methods may have parameters and return values; code for setting the in-parameters of these functions is generated automatically.

Silk Performer's Unit Import Tool offers you the option of selecting specific test case methods. It automatically generates BDL stub code (a benchmark description file) that calls those selected test case methods. Existing NUnit test classes can be called from Silk Performer without requiring modification of the test classes.

Note: Silk Performer's NUnit import utility supports NUnit versions 2.1.4 and higher.

Setting Up an NUnit or .NET Testing Silk Performer 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, expand **Unit Testing** and select **NUnit** or **.NET Testing**.
5. Click **Next**.

   Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.

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

Importing an NUnit or .NET Assembly

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **File** field, specify the archive that is to be tested. The archive is automatically added to the profile classpath. The available classes are then retrieved and displayed, sorted alphabetically in the **Class** field.
3. From the **Class** list, select one of the available classes for testing.

   When you do not specify a specific archive for testing, the **Model Script** dialog enables you to specify a class that is available in the profile classpath. Type the fully qualified class name into the **Class** field.

   The available constructors and methods are automatically retrieved and displayed.
4. From the **Constructor** list, select the appropriate constructor for instantiating the imported class.

   If a constructor is not selected (entry `<No Instance Required – Only Static Calls>`), only static methods will be displayed in the method list.
5. In the **Methods** area, select the methods that you want to call.
6. To filter the methods that are shown in the **Methods** area, perform the following steps:
   a) Click the **Advanced Settings** button (the funnel icon above the **Methods** area).
   b) Once you have customized filter settings, click **OK** to update the **Methods** area.
7. To change general NUnit settings including traffic redirection values such as routed web service proxy classes, click the **Active Profile Settings** link. The **Profile** dialog opens to the **General** page for .NET projects (NUnit and .NET Testing project types).

   Note: Changes made to these settings may lead to different results. Selections made in the **Class**, **Constructor**, and **Methods** fields will be updated with the new results.
8. Click **OK**.

Filter Options for NUnit Methods

This table lists the filters that are available in the **Methods** area when you import a test class.

NUnit Method	Description
Parameterless Functions Only	Ignore all functions that expect parameters

NUnit Method	Description
Unit Test Functions Only (selected by default)	Ignore all functions that are not NUnit
Member Functions Only	Ignore all non-member functions
Public Functions (selected by default)	Include public functions
Protected Functions	Include protected functions
Private Functions	Include functions with private access
Package Functions	Include functions with package access
Declared Functions Only	Ignore functions from the base classes
Complex Functions (selected by default)	Show functions that take complex parameters. In Java, complex parameters are scripted by `DotNetSetObject` with NULL as the default value.
Autodetect Unit Test Functions (selected by default)	Automatically detect and script functions. Functions must not have parameters or return values. Methods marked with `[SetUp]` and `[TearDown]` (for NUnit) respectively `[TestInitialize]` and `[TestCleanup]` (for Microsoft Unit Tests) are invoked prior to and following each method.

Example NUnit BDL Script

Script Example

The following script is generated by importing NUnit Money and selecting the three methods `BagMultiply`, `BagNegate`, and `BagSimpleAhdd`.

```
transaction TInit
var
  sFileName : string;
begin
  DotNetSetOption(DOTNET_OPT_REDIRECT_CONSOLE, 1);
  // ============================================
  // Unit Test TestClass Information:
  // Used Framework: NUnit Test Framework
  // Initialize method: SetUp
  // Class contains 21 test methods!
  // ============================================
  GetDataFilePath("nunitmoneysample.dll", sFileName);
  ghTestObj := DotNetUnitTestLoadObject(sFileName,
"NUnit.Samples.Money.MoneyTest", "NUnit.Samples.Money.MoneyTest");
end TInit;

transaction TMain
begin
  // BagMultiply
  DotNetUnitTestCallMethod(ghTestObj, "BagMultiply", "BagMultiply");
  // BagNegate
  DotNetUnitTestCallMethod(ghTestObj, "BagNegate", "BagNegate");
  // BagSimpleAdd
  DotNetUnitTestCallMethod(ghTestObj, "BagSimpleAdd", "BagSimpleAdd");
end TMain;

transaction TEnd
begin
  DotNetUnitTestFreeObject(ghTestObj, "NUnit.Samples.Money.MoneyTest");
end TEnd;
```

`DotNetUnitTestLoadObject` loads the NUnit assembly and creates an instance of the test class. If the test class has a global initialize function implemented (marked with TestFixtureSetup - ClassInitialize), this function will be called right after the object is created.

`DotNetUnitTestFreeObject` in the `TEnd` transaction frees the reference to the test object. If the test class has a global uninitialize function implemented (marked with `TestFixtureTearDown - ClassCleanup`), this function will be called prior to release of the object.

`DotNetUnitTestCallMethod` calls the `TestMethod` in the same way that the NUnit Microsoft Unit Test Engine does. If there is a `SetUp - TestInitialize` method implemented, it will be called before the test method is called.

If there is a `TearDown - TestCleanup` method implemented, it will be called after the test method is called.

Test methods that expect an exception to be thrown (those marked with `ExpectedException`) are only considered successful when an exception is actually thrown.

When a test method writes information to the error or out console, the information can be viewed in the TrueLog. A separate element is logged as child node of the `DotNetUnitTestCallMethod`.

The same applies for unexpected exceptions. Stack trace and exception messages are logged to the TrueLog.

Timers

A new optional timer parameter has been introduced for the original DotNet API calls and the new DotNetUnitTest API calls. When this parameter is specified, the execution times of the constructor, test method, set up method, and tear down method are measured. For the example above you would receive the following measures:

- For the constructor: `NUnit.Sample.Money.MoneyTest`
- For the methods: `BagMultiply`, `BagNegate`, `BagSimpleAdd`
- For the setup methods: `BagMultiply_Setup`, `BagNegate_Setup`, `BagSimpleAdd_Setup`

Silk Performer .NET Explorer

Silk Performer .NET Explorer, which was developed using .NET, and the Silk Performer .NET Framework allow you to test Web Services, .NET Remoting objects, and other GUI-less .NET objects. .NET Explorer allows you to define and execute complete test scenarios with different test cases without requiring manual programming because tasks are completed visually by way of mouse-based operations. Test scripts are visual and easy to understand, even for individuals who are unfamiliar with .NET programming languages.

Test scenarios created with .NET Explorer can be exported to Silk Performer for immediate reuse in concurrency and load testing and exported to Microsoft Visual Studio .NET for further customization.

For more information about .NET, visit *http://msdn.microsoft.com/net*.

Starting .NET Explorer

Perform one of the following steps to launch .NET Explorer:

- Click **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Development Tools** > **Silk Performer .NET Explorer** .
- Click **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Silk Performer Workbench** and create a new project with the application type `.NET/.NET Explorer` or `Web Services/.NET Explorer`.

By default, the executable file is located at `C:\Program Files\Silk\Silk Performer 20.0\DotNET Explorer\NetExplorer.exe`.

Refer to Silk Performer .NET Explorer Help for detailed information.

Using the .NET Explorer SilkEssential

Before you begin this task, create and save a .NET Explorer test scenario. Refer to Silk Performer .NET Explorer Help for detailed information.

Silk Performer ships with a SilkEssential that allows you to hook .NET Explorer to produce a BDL script with tested Web Services.

1. Create a new project based on the SilkEssential, as follows:
   a) 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.

   b) In the **Name** text box, enter a name for your project.
   c) Enter an optional project description in **Description**.
   d) From the **Application** menu tree, expand **Web Services** and click **.NET Explorer**.
   e) Click **Next**.

      Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.
2. Model the script.
3. While the recorder is running, start .NET Explorer.
4. Load a test scenario that you have defined in a previous.NET Explorer session.
5. Execute an animated run of the scenario. A BDL script is generated while the recorder and .NET Explorer run.
6. Close .NET Explorer and the recorder.
7. Perform a Try Script run of the recorded BDL script.
8. Perform XML customization and verification in the TrueLog, as required.

   Refer to TrueLog Explorer Help for detailed information.

Using the .NET Message Sample

This section includes concepts and tasks that describe the .NET message sample, which allows you to experiment with the testing of .NET technologies using Silk Performer.

The message sample implements a message server that allows users to log in, view their message store, and send messages to other users. The sample is comprised of the following .NET technologies:

- .NET Remoting
- Web services
- ASP.NET

.NET Message Sample Overview

The message sample implements a message server that allows users to log in, view their message store, and send messages to other users. The sample is comprised of the following .NET technologies:

- .NET Remoting
- Web services
- ASP.NET

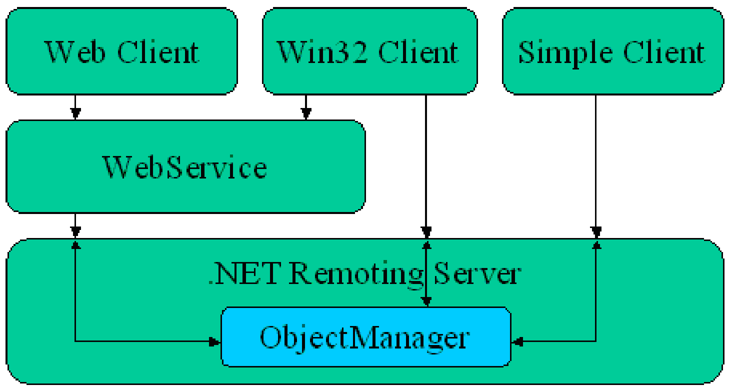
The sample contains the following components:

Component	Description
`MessageLib`	.NET library that defines the interfaces that are implemented by the server library.

Component	Description
`MessageImpl`	Implementation of the `MessageLib` interfaces.
`MessageServer`	A simple console application that hosts the .NET Remoting objects that are implemented in `MessageImpl`. An object that implements `IObjectManager` which can be activated at `http://localhost:1999/ObjectManager.rem`. You can change this setting in the `config` file.
`MessageServerIIS`	The `ObjectManager` remoting object hosted by IIS.
`MessageWebService`	This Web service exposes Web service methods for most of the methods implemented in the `MessageImpl`. The Web service does not host the objects of the `MessageImpl` directly. The Web service communicates with the .NET Remoting server.
`MessageSimpleClient`	This simple client console application acquires the `IObjectManager` from the server application and calls methods on the object.
`MessageWin32Client`	Win32 application that either connects to the .NET Remoting server or to the Web service. It provides an interface for logging in, checking messages, and sending messages.
`MessageWebClient`	ASP.NET application that connects to the Web service and provides an interface for logging in, checking messages, and sending messages.

Component Interaction Example

The .NET Remoting server hosts a Singleton SAO (Server Activated Object) that client applications (Web Service, Win32 Client, Simple Client) can aquire by way of `http://localhost:1999/ObjectManager.rem`.



MessageLib Interfaces

The `MessageLib` library defines several interfaces that are implemented by the `MessageImpl` library, which is hosted on the .NET Remoting server. `IObjectManager` is the core interface that can be acquired from the server. This interface offers access to other objects such as users, the message store, and messages. A separate interface assembly is required because when a client application accesses a remote object, it requires the type information of the object it acquires. By implementing a library that only defines the interfaces, the implementation remains on the server side. It would be possible to ship the `MessageImpl` to the client, but that would imply that the implementation exists on both the client and the server side, which is not the preferred approach.

Some methods return an integer value that indicates the success of the method call. `0` indicates a successful call. Values other than `0` indicate errors. The method `GetErrorMsg` on the `IObjectManager` interface returns an error description of the error code.

ObjectManager Interface

`Login` Method

```
int Login(string sUsername, string sPassword,
    ref IUser user)
```

This method executes a user login with a given password. If the login is successful, the user holds a valid reference to the user's object. If the login is not successful, check the return value.

`Logout` Method

```
int Logout(IUser user)
```

Logs out the user.

`CreateUser` Method

```
int CreateUser(string sUsername, string sPassword);
```

Creates a new user.

Note: This method does not login the user.

`DeleteUser` Method

```
int DeleteUser(IUser user);
```

Deletes the user. The user is no longer valid after this call.

`GetErrorMsg` Method

```
string GetErrorMsg(int errNo);
```

Returns a description for the error number.

IUser Interface

`ChangePassword` Method

```
int ChangePassword(string sOldPwd, string sNewPwd);
```

Changes the user password.

`ChangeName` Method

```
int ChangeName(string sNewName);
```

Changes the username.

`GetMessageStore` Method

```
IMessageStore GetMessageStore();
```

Returns the user's message store.

`SendMessage` Method

```
int SendMessage(string sRecipient, string sMsgText);
```

Sends a message to another user.

`Username` Method

```
string Username
```

User's name (read-only property).

IMessageStore Interface

`GetMessage` Method

```
int GetMessage(int nMsgId, ref IMessage msg);
```

Returns the message with the given message ID (1 based).

`DeleteMessage` Method

```
int DeleteMessage(int nMsgId);
```

Deletes the message with the given message ID (1 based).

`Count` Method

```
int Count
```

Number of messages in the store (read-only property)

IMessage Interface

`Sender` Method

```
string Sender
```

Sender (read-only property)

`MessageText` Method

```
string MessageText
```

Message text (read-only property).

Sample .NET Web Service

The sample Web service exposes most of the functions that are defined in the interfaces. When there is a successful login, the user object is stored in the ASP.NET session object. As a result, it is possible to call methods on the user object without passing a reference to the object back to the client of the Web service. The session state information is held in the server memory and the session key is passed back to the client in a cookie. So, if you are using a generated Web proxy client class, be sure to create a `CookieContainer` object and assign it to the `CookieContainer` property of the proxy object.

Exposed functions include:

- `Login`
- `Logout`
- `CreateUser`

- `DeleteUser`
- `GetMessageCount`
- `GetMessage`
- `DeleteMessage`
- `SendMessage`
- `GetErrorMsg`

The Web service defines a separate message class. The class contains two public members (`Sender` and `MessageText`).

Working With the .NET Message Sample

This section explains how to install and configure the .NET message sample. Also included is a detailed description of the clients and the server components. The section concludes with an explanation of how to test the .NET message sample with Silk Performer.

Installing the .NET Message Sample

Note: If you only need to test the .NET Remoting component of the message sample, skip this task and simply start the message server.

1. Create virtual directories for the Web service, Web client, and message server hosted on IIS by performing the following steps:
   a) Open the **Internet Service Manager** from the `Administrative Tools` folder.
   b) Create three virtual directories: `MessageWebService`, `MessageWebClient`, and `MessageServerIIS`.
   c) Open the sample application directory and set the base directory to the `MessageWebService`, `MessageWebClient`, and `MessageServerIIS` directories.

      By default, the sample application directory is `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample`.
   d) Right-click the virtual directory entry in the tree and choose **Properties**. The **Properties** window opens.
   e) Click **Create** to create an application for the virtual directory.

      If there is no **Create** button on the **Properties** page, this step has been executed automatically by IIS during the creation of the virtual directory.
   f) Repeat the preceding two steps ('d' and 'e') for both virtual directories.
2. To ensure that the binaries of the sample application are compiled against the latest version of the .NET Framework, perform the following steps:
   a) Open the `MessageLib.sln` solution in the `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\MessageLib` directory.
   b) Perform either a release or debug build of the solution.
   c) Copy the output files of the `MessageImpl` (either debug or release) project to the `MessageServerIIS\bin` directory.

Configuring the .NET Message Sample

Configuring the IObjectManager Object Properties

By default the message server exposes the `IObjectManager` object on `http://localhost:1999/ObjectManager.rem`. Proceed with this task only if you need to change the default setting.

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\MessageServer` and double-click `MessageServer.exe.config`.

2. Enter the port number.

   Ensure that the port is not already in use by another service on your machine.
3. To use a TCP channel instead of `HTTP`, change the `ref` attribute to `TCP`.
4. To change the URL of the object, edit the `objectUri` attribute.

If you change any of the these properties, ensure that you also change the configuration of the client applications, as explained in the following tasks.

Configuring the Simple Client

If you changed the `ObjectManagerURL` value in `MessageServer.exe.config`, then you also need to edit the simple client config file.

1. Navigate to `<installation path>\Sample Apps\Message Sample\MessageSimpleClient`.
2. Double-click `MessageSimpleClient.exe.config`.
3. Change the value of the key `ObjectManagerURL` to the required URL.

Configuring the Web Service

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET \Message Sample\MessageWebService` and double-click `Web.config`.
2. Change the value of the key `ObjectManagerURL` to the required URL.

You can now access the Web service at `http://<server>/MessageWebService/ MessageService.asmx`.

Configuring the Web Client

When the Web service is installed on a different machine than the Web client, or if you have selected a different virtual directory name, you must specify the URL of the Web service in the file `Global.asax`.

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET \Message Sample\MessageWebService` and double-click `Global.asax.cs`.
2. Locate the `Session_Start` method.
3. Define your Web service's URL.
4. Rebuild the application.

You can now access the Web client at `http://<server>/MessageWebClient/Default.aspx`.

Exploring the .NET Message Sample

This section guides you through the components of the message sample. The sample uses the message server that is implemented by the console application. You can use the message server running in IIS by changing the remoting URL in the configuration files of the client applications.

Starting the .NET Remoting Server

The .NET Remoting server hosts the `ObjectManager` object that is accessed by all client applications. The server is a simple console application that runs when you press any key. Some of the methods of the remoting objects print information to the console, such as which users have logged in or out, and who sent messages to whom.

Note: The server holds all the information about the users and messages in memory. So if you stop the server, the current state is lost. This means that each time you start the server you have an empty environment with no users or messages.

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\MessageServer`.
2. Double-click `MessageServer.exe`.

The message server console window displays. The console window logs information about the methods that are called during tests. Press any key to stop the server.

Running the Simple Client

The simple client connects to the .NET Remoting server, creates two users (`User1` and `User2`) with passwords (`Pass1` and `Pass2`). The simple client also sends a message from `User1` to `User2`.

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\MessageSimpleClient`.
2. Double-click `MessageSimpleClient.exe`.

The message server console window opens. Two users are created and a message is sent from `User1` to `User2`.

Working With the Win32 Client

Before working with the Win32 client, run the simple client. The simple client automatically creates system users with which the Win32 client can interact.

The Win32 client does not require any configuration outside the application.

The Win32 client can either connect directly to the `ObjectManager` object of the remoting server or it can perform all the method calls through the Web service. The Web service forwards the method calls to the `ObjectManager` object on the remoting server.

Once you connect to the Win32 client, you can either create a new user or log in as an existing user. You can also delete the current user. Any messages sent to the active user are listed in the message store, including any messages sent using the sample Win32 client or the simple client. You can delete messages or send new messages to other users in the system.

The **Last Error** box shows the error text of the last error that occurred when a method was executed.

Note: When errors occur while calling methods on remote objects, you see error descriptions in the **Error Text** box.

1. Navigate to `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\MessageWin32Client` and double-click `MessageWin32Client.exe`. The sample Win32 client starts.
2. Connect to either the Remoting server or the Web service.
   - To log into the .NET Remoting message server, enter the server's URL (for example, `http://localhost:1999/ObjectManager.rem`) in the **Message Server** field and click **Connect**.
   - To log into the sample Web service, enter the Web service URL (for example, `http://localhost/MessageWebService/MessageService.asmx`) in the **Web Server** field and click **Connect**.
3. Login using one of the users that the simple client application created.

   By default, the simple client application creates `User1` with password `Pass1` and `User2` with password `Pass2`.
4. Create a new user.
5. Login using the newly created login credentials.
6. Send a message to another user.

Note: User names are case-sensitive.

7. Review the server console output.

   Logging information related to all methods that were called is listed there.

Be sure to log out after you finish your work.

Creating a New User

1. Enter unique **Username** and **Password** values.
2. Click **Create User**.

Sending a Message

1. Enter an existing username in the **To User** field.
2. Enter sample text in the **Message** field.
3. Click **Send**.

Working with the Web Client

Before you can use the Web client, you must configure the Web client by specifying the URL of the Web service.

Before working with the Web client, run the simple client. The simple client automatically creates system users with which the Web client can interact.

The Web client communicates with the Web service. The Web client allows you to login to the system and create new users. When you login, you can view your message store, delete your messages, or send new messages to other users in the system. Usernames are case-sensitive. If an error occurs while you are executing a method, the error will be displayed in red.

1. Start your browser and load the URL `http://localhost/MessageWebClient/default.aspx`. A login page is displayed.
2. Enter login credentials for an existing user account or create a new user account (see instructions in the task below).

   The first time you login, use `User1` for the **Username** and `Pass1` for the **Password**.
3. Click **Login**. When you log in you are redirected to the **Your Message Store** page where you can see any messages that are in the active user's message store, including any messages that have been sent using the sample Win32 client or the simple client.

   You can send messages to other users in the system, same as you can with the sample Win32 client (see instructions in the task below).
4. Review the server console output window.

   The window displays information related to the activities you performed with the Web client, including user creation, sent messages, and logins.

Be sure to log out after you finish your work.

Creating a New User

1. Enter unique **Username** and **Password** values.
2. Click **Create User**.

Sending a Message

1. Enter an existing username in the **To User** field.

2. Enter sample text in the **Message** field.
3. Click **Send**.

Testing the .NET Message Sample

You have several options for testing the .NET message sample with Silk Performer:

- Hook into the Win32 client application and record Web traffic.
- Record a Web session with the Web client.
- Use .NET Explorer to test the Web service and generate a Silk Performer .NET project.
- Create a new .NET Framework project and write a .NET test driver that either tests the Web service or the .NET Remoting server directly.

Hooking a Win32 Client Application

Before proceeding, start the message server.

1. Create a new Silk Performer project of type **Web Services** > **XML/SOAP**.
2. Add a new recording profile for `MessageWin32Client.exe`.
   a) From the Silk Performer **Application type** list box, select **Custom Application**.
   b) In the **Protocol selection** area, check the **Web** check box. The Silk Performer Recorder intercepts all function calls and displays the results.
3. Begin recording a script.
4. Perform actions using MessageWin32Client (for example, connect, login, send, and logout).
5. End recording.

   For each call made in the Win32 client application there is a `WebUrlPostBin` call and a SOAP envelope.
6. Review the recorded script.
7. Execute a Try Script run using the recorded script.

   Monitor details of the test run in the **Server Console** window.

Recording a Web-Client Session

.NET Explorer

Before proceeding, start the message server.

1. Create a new Silk Performer project of type **Web Browser** > **Web business transaction**.
2. Begin recording: `http://localhost/MessageWebClient/Default.aspx`.
3. Perform actions using the sample Web client (for example, connect, login, send, and logout).
4. End script recording.
5. Review the recorded script.
6. Execute a Try Script run using the recorded script.

   Monitor details of the test run in the **Server Console** window.

Testing a .NET Web Service

Before proceeding, start the message server.

Note: This scenario is stored within your Silk Performer installation. You can access it at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample \NetExplorer_TestingWebService`.

1. Launch Silk Performer and create a new project.

2. Load the WSDL from the Web service: `http://localhost/MessageWebService/MessageService.asmx?WSDL`
3. Invoke the `login` method using `User1` and `Pass1` as input values.

   You can also use random variables or global variables mapped to attributes (see .NET Explorer Help for details).
4. Send a message with some text from `User1`.
5. Logout the current user and login as `User2` (username: `User2`; password: `Pass2`).
6. Invoke `GetMessageCount` and store the result in a variable called `mCount`.
7. Call `GetMessage` with the message count variable as input. You will receive the last message that was sent.
8. Add the remaining logout method call.
9. Execute an animated run.

   The failed verification can be ignored (or you can remove the default verification of the `GetMessageCount` call).
10. Export a Silk Performer .NET project or a Visual Studio .NET project (exported projects can be found at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\NetExplorerExportedProject`).
11. Execute a Try Script run via Silk Performer or Visual Studio .NET.
12. Explore the TrueLog. You will see a node for each Web service call.

Creating a .NET Framework Project

Before proceeding, start the message server.

For full details about the Silk Performer .NET Framework, see *.NET Framework Help.*

1. Create a new Silk Performer project of type **.NET** > **.NET Framework using Visual Studio .NET Add-On**.
2. Use the **Model Script** dialog to bring up Visual Studio .NET.
3. Code a .NET test driver.
4. A sample test driver implementation that tests the .NET remoting server and the Web service can be found at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\Message Sample\SPVNetTestDriver`.
5. Open the project in Visual Studio .NET and execute a Try Script run.
6. After the run, review the TrueLog and open the user's WRT file.

The sample sends a message from `User1` to `User2` (where `User1` is connected to the remoting server and `User2` is connected to the Web service).

.NET Explorer Samples

This section includes topics that explain how to access the .NET Remoting and Web service samples that ship with .NET Explorer. It also includes a list of helpful Web resources.

.NET Remoting Support

`RemotingLib` implements one remotable object (`MarshalByRefObject`) and defines an interface. The interface is available at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\RemotingSamples\RemotingLib`.

`RemServerConsole` is the server application that hosts the remotable object and implements one object that implements the defined interface of `RemotingLib`. It is available at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\RemotingSamples\RemServerConsole`.

All source code is available. Code is already compiled into a debug version.

.NET Remoting Testing

1. Start the remoting server application via .NET Explorer. Choose **Help** > **Start Remoting Sample**.
2. Manually start `RemServConsole.exe`, which is available at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\RemotingSamples\RemServerConsole\bin\debug\RemServConsole.exe`.
3. Load `RemotingLib.dll`, which is available at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\RemotingSamples\RemServerConsole\bin\debug\RemotingLib.dll`.
4. Create an instance of the class `RemoteObject`. Right-click `RemoteObject` and choose **Create Remote Object**. You will be prompted for the activation URL. Enter `tcp://localhost:2000`.
5. Connect to the remoted interface.
6. Invoke a method of the `RemoteInterface` interface. The first time you do this you will be prompted for the activation URL. The activation URL is either `tcp://localhost:2000/RemoteInterface.rem` or `tcp://localhost:2000/RemoteInterfaceSCall.rem` (single call object).

.NET Component Testing Overview

`netexptestdll.dll` implements base classes that can be tested. This file is available at `<public user documents>\Silk Performer 20.0\SampleApps\DOTNET\netexptestdll\bin\debug\netexptestdll.dll`.

You can instantiate one of the loaded objects via the context menu on the class or by invoking a constructor. Following that, store the created object in a variable and call methods on the created object.

Sample Web Services

Several sample Web services are available for testing with .NET Explorer.

- The .NET sample Web service `SampleService.asmx` offers simple method calls.
- Testing ASP.NET Data Types (`DataTypes.asmx`) is a sample Web service provided by Microsoft that allows you to test different data types that are provided by the ASP.NET Framework.
- Order Web service (`OrderService.asmx`) is a sample Web service that allows you to simulate the ordering of books and DVDs.
- Other free accessible Web services are available at *http://www.xmethods.net* and *http://www.mssoapinterop.org*.

Order Web Service

The sample Order Web service offers a number of methods and is available for testing with .NET Explorer at *http://demo.borland.com/OrderWebService/OrderService.asmx*. The following methods are available:

- `SearchArticles`: Searches for articles by a name pattern.
- `GetArticleByName`: Searches for an article by the full name.
- `GetArticleById`: Searches for an article by its ID.
- `CreateUser`: Creates a new user in the database.
- `Login`: Logs in an existing user. The return value is the user ID that is needed for other method calls (store this return value in a global variable).
- `Logout`: Logs the user out of the system.
- `CreateOrder`: Creates a new order for the currently logged in user.
- `AddOrderItem`: Adds an item to the order.
- `GetOrder*`: Returns information about an order.

All order related methods take the `userid` as a header parameter. This is why you should store the user ID that is returned by the `Login` method in a global variable and use the variable as input for all order-related methods.

Some of the methods may throw SOAP exceptions (for example, if you are not logged in and try to create an order).

Web References

Here are some additional online resources that can assist you with the testing of Web services and getting the most from .NET Explorer.

- (Web service) *http://demo.borland.com*
- (Web service) *http://www.mssoapinterop.org/asmx/simple.asmx*
- (Web service) *http://www.mssoapinterop.org/asmx/simpleB.asmx*
- (.NET resource) *http://msdn.microsoft.com/net*

GUI-Level Testing Support

When to Use GUI-Level Testing

Suppose you have an application that implements a traditional client/server architecture. You want to test what happens when multiple instances of the client application access the server simultaneously. An example would be a proprietary time-tracking system that stores the working hours of employees on a server. However, you cannot use any of the existing Silk Performer application types for testing, because the application uses an exotic protocol to communicate between client and server. In such instances, you may want to use GUI-level testing. With GUI-level testing, you can find out how many employees can concurrently access the server and use the time-tracking system without overloading the server.

How GUI-Level Testing Works

As shown in the graphic below, the process begins with the Silk Performer Controller. When you start a load test, the Silk Performer Controller connects to an agent running on a Microsoft Windows Server operating system and has *Remote Desktop Service* (formerly known as *Terminal Services*) running. The Silk Performer agent then creates runtimes and within the runtimes the virtual users are created. Each virtual user creates a new session on the agent machine and starts Silk Test within this session. Silk Test then performs the previously recorded steps on the application, or in other words: Silk Test drives the application. The application accesses the server and generates the load, while the virtual users measure the response times.

Sometimes you cannot run more than one instance of an application on a machine or within a single session. This is why each virtual user creates a new session on the agent machine. With this approach, Silk Performer creates several instances of your application, all of which access the system under test simultaneously.

Setting Up a GUI-Level Testing Environment

1. Set up at least one agent machine that has one of the Microsoft Windows Server editions installed on it.
2. Install Silk Performer Agent on the machine.
3. Install Silk Test.
4. Install your client application.
5. Use Silk Test to model one or more test cases using the application.
6. Create a Silk Performer GUI-level testing project that uses the Silk Test project to run the defined test cases against the system under test.

Once you have performed all these steps, you can start the load test in Silk Performer.

Note: You can use the following Silk Test clients for GUI-level testing: Silk Test Classic, Silk4J, and Silk4NET. Make sure that you meet all requirements when you use Silk4J and Silk4NET for GUI-level testing. See *Requirements for GUI-Level Testing with Silk4J and Silk4NET* for details.

Why is it Called *GUI-Level Testing*?

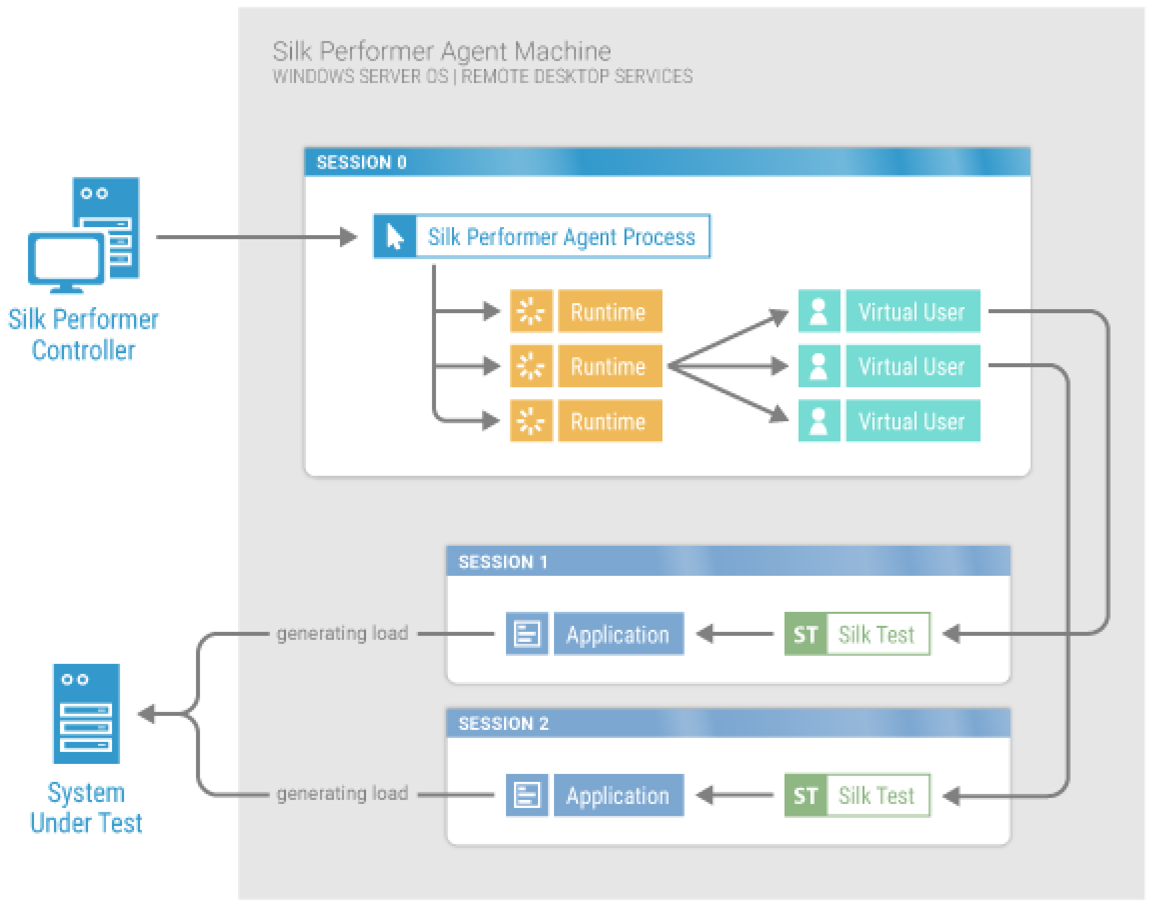
Silk Test performs testing directly on the graphical user interface, or in other words, on the GUI-level. With this approach you can watch how Silk Test performs the recorded test steps, for example mouse clicks and keyboard entries, if you connected to one of the sessions on the agent machine.

GUI-Level Testing Functions

Refer to the Silk Performer BDL Reference for full details on the BDL functions that are offered by Silk Performer.

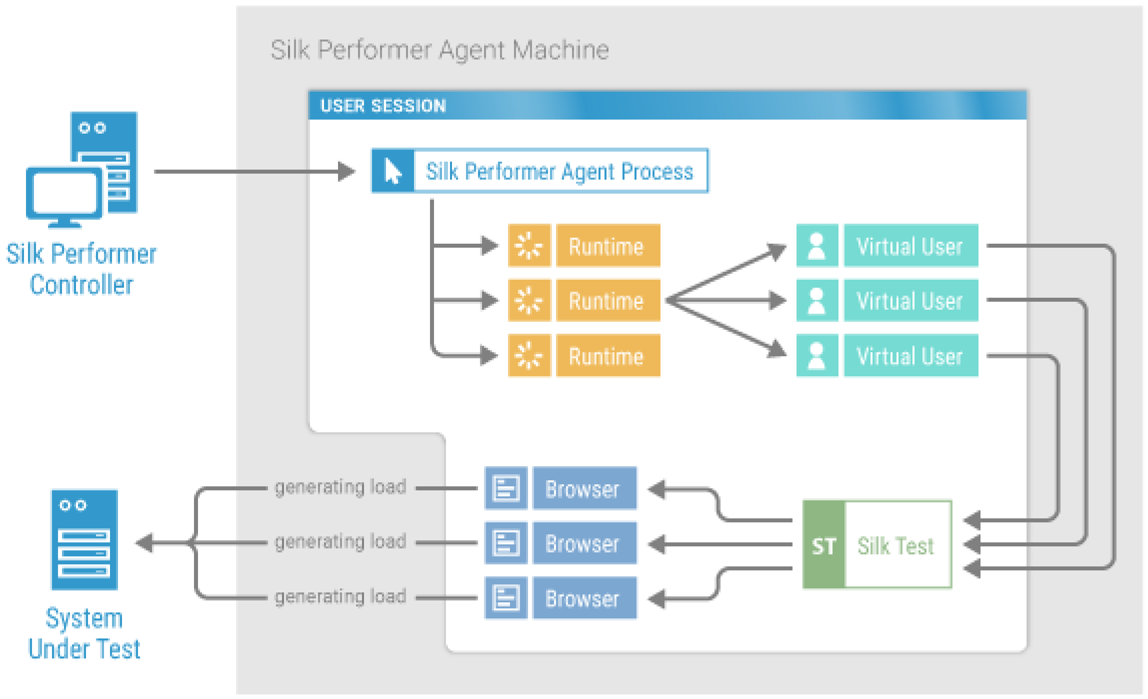
Note: Silk Test can be started in local host mode. With this approach, virtual users use a console session rather than a separate Windows session.

Note: GUI-level testing only works on Windows Server edition platforms. Windows Home or Professional editions can only be used as agents for a single GUI-level testing VUser.



Single Session GUI-Level Testing

For tests against web applications using Google Chrome or Mozilla Firefox, Silk Performer allows you to run all virtual users within a single Windows session. The benefits are that no remote desktop licenses are required and that resource consumption per virtual user is considerably lower compared to the conventional GUI-level testing approach.



Configuring Windows for GUI-Level Testing

Before you can execute GUI-level tests, you must configure your Windows operating system. Additionally, Silk Test needs to be installed on the agent computer (refer to the Silk Test Help for details).

Configuring Windows 2008 R2 for GUI-Level Testing

Before you can perform this task, make sure that the **Remote Desktop Services** server role is installed.

1. Enable Remote Desktop Protocol (RDP).

   RDP is disabled by default.

   a) Open Windows **Control Panel** > **System and Security** > **System**.
   b) Click the **Remote Settings** link.
   c) Check the checkbox **Allow connections from computers running any version of Remote Desktop (less secure)**.
   d) Click **OK**.
2. Allow RDP users to launch applications remotely.
   a) Navigate to **Administrative Tools** > **Remote Desktop Services** > **RemoteApp Manager**.
   b) Click **Change** next to **RD Session Host Server Settings**.
   c) In the **Access to unlisted programs** group box, check the checkbox **Allow users to start both listed and unlisted programs on initial connection**.
   d) Click **OK**.
3. Allow RDP users to run multiple sessions.
   a) Navigate to **Administrative Tools** > **Remote Desktop Services** > **Remote Desktop Session Host Configuration**.
   b) Double-click **Restrict each user to a single session**. The **Properties** dialog box displays.
   c) Uncheck the checkbox **Restrict each user to a single session**.
   d) Click **OK**.
4. Configure Remote Desktop settings.

a) Navigate to **Administrative Tools** > **Remote Desktop Services** > **Remote Desktop Session Host Configuration**.
b) Right-click **Remote Desktop Protocol-TCP (RDP-Tcp)** in the **Connections** list and click **Properties**.
c) Click the **Log on Settings** tab and make sure that **Always prompt for password** is disabled.
d) Click the **Sessions** tab and make sure that **Override user settings** is selected and that **End a disconnected session** is set to `1 minute`. Make sure that all other settings are disabled or left blank.
e) Click the **Environment** tab and make sure that **Run initial program specified by user profile and Remote Desktop Connection or client** is enabled.
f) Click the **Remote Control** tab and make sure that **Use remote control with default user settings** is enabled.
g) Click the **Network Adapter** tab and make sure that **All Network adapters configured with this protocol** is selected in the **Network adapter** list.
h) Click **OK**.

5. User Account Control (UAC) is enabled by default, but is not required for GUI-level testing. If you want to leave UAC turned on, the agent must run under a user account.

   To configure UAC settings:

   a) Navigate to **Control Panel** > **User Accounts** > **User Accounts** > **Change User Account Control Settings**.
   b) Modify the UAC notification level as desired.
   c) Click **OK**.
6. Using the Windows user and group administration functionality, select the local users that can execute GUI-level tests. Ensure that this user is a member of the `Administrators` and/or `Remote Desktop Users` group.

Configuring Windows 2012 - 2019 for GUI-Level Testing

Before you can perform this task, make sure that **Remote Desktop Services** is enabled and that you are logged in with a domain user with administrative privileges on the machine.

1. Set a time limit for disconnected users.

   Note: The time limit can be set on two levels: either through a Windows group policy or through the Remote Desktop Services Collection. The group policy setting has priority over the Remote Desktop Services Collection setting.

   Group Policy:

   a) Start the **Windows Local Group Policy Editor** and navigate to **Local Computer Policy** > **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Remote Desktop Services** > **Remote Desktop Session Host** > **Session Time Limits**.
   b) Double-click **Set time limit for disconnected sessions.**
   c) Click **Enabled**.
   d) For **End a disconnected session**, select 1 minute.

   Remote Desktop Services Collection:

   a) Start the **Windows Server Manager** and navigate to **Remote Desktop Services** > **Collections** > **<name of the collection>**. In the **Properties** area, select **Edit Properties** from the **Tasks** menu.
   b) On the **Session Collection** dialog, select **Session**.
   c) For **End a disconnected session**, select 1 minute.
   d) Click **OK**.
2. Allow RDP users to run multiple sessions and launch all programs.

   a) Start the Windows **Local Group Policy Editor** and navigate to **Local Computer Policy** > **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Remote Desktop Services** > **Remote Desktop Session Host** > **Connections**.

   b) Double-click **Restrict Remote Desktop Services users to a single Remote Desktop Services session**.
   c) Click the **Disabled** option.
   d) Click **OK**.
   e) Double-click **Allow remote start of unlisted programs**.
   f) Click the **Enabled** option.
   g) Click **OK**.
3. User Account Control (UAC) is enabled by default, but is not required for GUI-level testing. If you want to leave UAC turned on, the agent must run under a user account.

   To configure UAC settings:

   a) Navigate to **Control Panel** > **User Accounts** > **User Accounts** > **Change User Account Control Settings**.
   b) Modify the UAC notification level as desired.
   c) Click **OK**.
4. Add users to the `Remote Desktop Users` group.
   a) In **Server Manager** > **Tools** > **Computer Management** > **Local Users and Groups** > **Groups**, double-click the **Remote Desktop Users** group and add the local users that shall be able to execute GUI-level tests.
   b) In case the GUI-level test users require administrative privileges during test execution, you can add them to the **Administrators** group here.

Obtaining More Licenses for Windows Server Edition

Windows Server Edition offers multiple licenses for remote desktop sessions, which means that you can connect multiple times during a single session.

Windows Server Edition offers two free licenses for remote desktop sessions. Windows Home and Professional editions can only be used as agents for a single GUI-level testing VUser.

1. Navigate to **Start** > **...** > **Control Panel** > **Add or Remove Programs** .
2. Click the **Add/Remove Windows Components** button.
3. Within the **Windows Components Wizard**, scroll to and check the **Terminal Server** check box.
4. Click **Next** to enable application mode for your Windows system.

   Note: A license server is required for application mode. Contact Microsoft for information about obtaining a license server.
5. Restart your computer.

GUI-Level Test Execution

Modeling GUI-Level Tests - Keyword-driven

1. Click **File** in the menu and click **New Project**. In the tree, click **GUI-Level Testing** and **Silk Test**. Enter a **Name** and a **Description** and click **Next**.
2. In the **File** field, specify the Silk Test asset you want to use for a performance test. Silk Performer automatically detects the file type and enables the appropriate button below.
3. Click **Import Keyword-Driven test**.
4. In the **Tests** area, select the keyword-driven tests you want to import.
5. Specify the **Import Options**:

   - **Ignore parameters**: Does not include any of the parameters that are part of the keyword-driven test.
   - **Script parameters as static strings**: The parameters are directly added to the script, which means that they are static strings within the script.

- **Script parameters as values from CSV file**: The parameters are stored within a .csv file and referenced in the script. The .csv file is added to the project and displays within the **Project** tree.

6. Specify the **Row selection order**:
   - **Random**: Adds the BDL function `FileGetRndRow` to the script.
   - **Sequential - machine-wide**: Adds the BDL function `FileGetNextRow` to the script.
   - **Sequential - test-wide**: Adds the BDL function `FileGetNextUniqueRow` to the script.
7. Select a **Web Browser** from the drop-down list. This list is only enabled when the file that is to be imported is based on a Silk Test web project. Silk Test web projects can make use of the single-session concept for GUI-level testing.
8. Enable **Use project attributes for session login** to let Silk Performer use credentials from the **Project Attributes** to login into sessions. To edit the project attributes, click **Project** > **Project Attributes** . The credentials will be added to the `TInit` transaction of your script.

   Note: Silk Test web projects can make use of the single-session concept for GUI-level testing, thus login credentials are not required at all. Nevertheless, for further script customization it might be useful to enable **Use project attributes for session login**.
9. Click **OK** and save the .bdf file.

Silk Performer imports the test assets and generates an appropriate .bdl stub.

Modeling GUI-Level Tests - Silk4J

1. Click **File** in the menu and click **New Project**. In the tree, click **GUI-Level Testing** and **Silk Test**. Enter a **Name** and a **Description** and click **Next**.
2. In the **File** field, specify the Silk Test asset you want to use for a performance test. Silk Performer automatically detects the file type and enables the appropriate button below.
3. Click **Import Silk4J test**.
4. In the **File** field, specify the archive that is to be tested. The archive is automatically added to the profile classpath. The available classes are then retrieved and displayed, sorted alphabetically in the **Class** field.
5. From the **Class** list, select one of the available classes for testing.

   When you do not specify a specific archive for testing, the wizard enables you to specify a class that is available in the profile classpath. Type the fully qualified class name into the **Class** field, for example `java.lang.String`.

   The available constructors and methods are automatically retrieved and displayed.
6. In the **Methods** area, select the methods that you want to call.
7. To filter the methods that are shown in the **Methods** area, perform the following steps:
   a) Click the **Advanced Settings** button (the funnel icon above the **Methods** area).
   b) Once you have customized filter settings, click **OK** to update the **Methods** area.
8. To change general Java settings including the Java version, Java home directory, or JVM DLL, click the **Active Profile Settings** link. The **Profile Settings** dialog opens to the **Java/General** page for Java projects (JUnit project type).

   Note: Changes made to these settings (for example Java Classpath) may lead to different results. Selections made in the **Class**, **Constructor**, and **Methods** fields will be updated with the new results.

   Note: If you change the Java version, Java home directory, or JVM DLL, you must restart Silk Performer for the changes to take effect.
9. Select a **Web Browser** from the drop-down list. This list is only enabled when the file that is to be imported is based on a Silk Test web project. Silk Test web projects can make use of the single-session concept for GUI-level testing.

10. Enable **Use project attributes for session login** to let Silk Performer use credentials from the **Project Attributes** to login into sessions. To edit the project attributes, click **Project** > **Project Attributes** . The credentials will be added to the `TInit` transaction of your script.

    Note: Silk Test web projects can make use of the single-session concept for GUI-level testing, thus login credentials are not required at all. Nevertheless, for further script customization it might be useful to enable **Use project attributes for session login**.
11. Click **OK** and save the .bdf file.

Silk Performer imports the test assets and generates an appropriate .bdl stub.

Modeling GUI-Level Tests - Silk4NET

1. Click **File** in the menu and click **New Project**. In the tree, click **GUI-Level Testing** and **Silk Test**. Enter a **Name** and a **Description** and click **Next**.
2. In the **File** field, specify the Silk Test asset you want to use for a performance test. Silk Performer automatically detects the file type and enables the appropriate button below.
3. Click **Import Silk4NET test**.
4. In the **File** field, specify the archive that is to be tested. The available classes are retrieved and displayed, sorted alphabetically in the **Class** field.
5. From the **Class** list, select one of the available classes for testing. Type the fully qualified class name into the **Class** field. The available methods are automatically retrieved and displayed.
6. In the **Methods** area, select the methods that you want to call.
7. To filter the methods that are shown in the **Methods** area, perform the following steps:
    a) Click the **Advanced Settings** button (the funnel icon above the **Methods** area).
    b) Once you have customized filter settings, click **OK** to update the **Methods** area.
8. To change general .NET settings, click the **Active Profile Settings** link. The **Profile Settings** dialog opens to the **.NET/General** page.
9. Select a **Web Browser** from the drop-down list. This list is only enabled when the file that is to be imported is based on a Silk Test web project. Silk Test web projects can make use of the single-session concept for GUI-level testing.
10. Enable **Use project attributes for session login** to let Silk Performer use credentials from the **Project Attributes** to login into sessions. To edit the project attributes, click **Project** > **Project Attributes** . The credentials will be added to the `TInit` transaction of your script.

    Note: Silk Test web projects can make use of the single-session concept for GUI-level testing, thus login credentials are not required at all. Nevertheless, for further script customization it might be useful to enable **Use project attributes for session login**.
11. Click **OK** and save the .bdf file.

Silk Performer imports the test assets and generates an appropriate .bdl stub.

Modeling GUI-Level Tests - Silk Test Classic

1. Click **File** in the menu and click **New Project**. In the tree, click **GUI-Level Testing** and **Silk Test**. Enter a **Name** and a **Description** and click **Next**.
2. In the **File** field, specify the Silk Test asset you want to use for a performance test. Silk Performer automatically detects the file type and enables the appropriate button below.
3. Click **Import Silk Test Classic test**.
4. If the test case file you want to import is located within a Silk Test package file (.stp), select **Open a Silk Test package file** and specify the file in the **Silk Test Package** field.
5. If you want to import a test case file, select **Open a Silk Test script file** and specify the file in the **Script File** field.

6. Select a specific **Testcase** from the list.
7. *(optional)* You can add Silk Test Classic test data to the selected test case, if required. Enter test data into the **Test Data** field using the format `"<test case name>",<test data>` (For example, `"test", 10`).
8. Click **Add**. The selected test case appears below in the **Testcase** field.
9. Add more test cases to your project as required by repeating this procedure.
10. Select a **Web Browser** from the drop-down list. This list is only enabled when the file that is to be imported is based on a Silk Test web project. Silk Test web projects can make use of the single-session concept for GUI-level testing.
11. Enable **Use project attributes for session login** to let Silk Performer use credentials from the **Project Attributes** to login into sessions. To edit the project attributes, click **Project** > **Project Attributes** . The credentials will be added to the `TInit` transaction of your script.

    Note: Silk Test web projects can make use of the single-session concept for GUI-level testing, thus login credentials are not required at all. Nevertheless, for further script customization it might be useful to enable **Use project attributes for session login**.
12. Click **OK** and save the .bdf file.

Silk Performer imports the test assets and generates an appropriate .bdl stub.

Exporting Silk Test Tests to Silk Performer

Before you start the export, make sure that Silk Performer is installed on your computer.

1. You can export your Silk Test test from Eclipse and Visual Studio:
    - Start Eclipse and open or create a Silk4J test.
    - Start Visual Studio and open or create a Silk4NET test.
2. Use the **Export as Performance Tests** feature:
    - In Eclipse, click the Silk Test icon in the toolbar and click **Export as Performance Tests**.
    - In Visual Studio, click **Silk4NET** in the menu and click **Export as Performance Tests**.

Silk Performer launches and the **Model Script** dialog displays. Set the required options and complete the import process. Silk Performer then creates a .bdf file and adds all necessary files to the project.

User Credentials for GUI-Level Testing

User credentials for GUI-level testing can be specified in the following areas:

- Profile settings
- Project attributes (username and password project attributes are automatically defined when you create a GUI-level testing project)
- Plain text specified in the BDL script
- Imported from data files

Note: Ensure that the user accounts used for GUI-level testing are members of the Remote Desktop Users Windows group on the remote agent.

If you want each VUser to connect using different login credentials, specify the credentials using project attributes or use script customization through data files.

If you want each VUser to connect with identical login credentials, specify the credentials using profile settings or with plain text in the BDL script.

Note: User credentials specified in profile settings are used only when the other options listed above are not used. When no user credentials are specified in any of the areas listed above, Silk Performer connects to the console session without using the remote desktop protocol. In such instances you can only run one VUser per agent.

Timers in GUI-Level Testing

Timers are central to GUI-level load testing. You can add timers to your Silk Test Classic, Silk4J, and Silk4NET scripts which will be reported to Silk Performer's test results. Refer to the Silk Test Help for detailed information about creating timers within Silk Test scripts.

Silk Performer automatically generates names for Silk Test timers that do not have names.

When executing keyword-driven tests, the execution time for each keyword is logged automatically.

GUI-Level Testing Result Files

You can find the most recent Try Script TrueLog files in the `RecentTryScriptTest` directory within your Silk Performer project directory. During GUI-level testing, temporary Silk Test TrueLog files with the extension `.xlgs` are written. After each Silk Test test case execution, the results of the Silk Test `.xlgs` and the results of the Silk Performer `.xlgs` files are merged into the Silk Performer `.xlgs` files (per VUser) and the temporary `.xlgs` files are deleted.

The `RecentTryScriptTest` directory within your Silk Performer project directory also includes Silk Test `.xlgs` result files. These are the files that are displayed in Silk Test when you initiate the **Explore** Silk Test **results** command.

Exploring Silk Test Results

1. Within Silk Performer, right-click a virtual user profile.
2. Select **Explore** Silk Test **Results** from the context menu. Silk Test launches, allowing you to analyze the corresponding Silk Test `.res` result file.

   You can also select **Explore TrueLog** from the context menu to view a Try Script's TrueLog in TrueLog Explorer.

   Click the **Results** tab to view test results directly in Silk Performer.

Requirements for GUI-Level Testing with Silk4J and Silk4NET

Make sure to meet the following requirement when you use Silk4J for GUI-Level testing:

- You must have Silk Test 15.0 or higher installed.

Make sure to meet the following requirements when you use Silk4NET for GUI-Level testing:

- You must have Silk Test 15.0 or higher installed.
- You must have Test Agent from the .NET Framework installed.
- You need the same version of MSTest that was used to build the test file.

  Note: Silk Performer will always use the latest MSTest version that is installed on the test machine. If the version you used for building the test file differs from the latest version that is installed on the test machine, the Silk4NET information in the TrueLog file will be missing.

GUI-Level Testing Scalability Scenarios

System Test Environment

- Silk Performer 2008
- Silk Test 2008
- 4 GB RAM
- BroadCom 57xx Gigabit Controller

- Dual (2 x 2.4 GHz) processors
- 167 GB free space hard disk

Note: The results below should be used as a guideline only and your results may differ depending on the application under test and the available resources of specific Silk Performer agent machines.

Test Scenarios

Executing a 1 VUser test over a 10 minute period

- 40k of Memory for `Performer.exe`
- 7.2k of Memory for the `PerfRun.exe` (Silk Performer replay engine)

During the test, the VUser will launch a terminal service session which in turns then launches the following processes (this scenario is true for every VUser running a test).

- Windows terminal services = 6.3k
- `Partner.exe` (Silk Test) = 21k
- `Agent.exe` (Silk Test Agent) = 8k
- `Perfsm.exe` (Silk Performer Session Manager) = 1.09k
- `Testapp.exe` (application under test) = 6.5k (this is the only real variable within the terminal service session in a load test scenario)

Executing a 5 VUser increasing workload test over a 10 minute period, with VUser per process set to 1

- CPU% = anywhere between 1% to 51% depending on which state the Silk Test process was in while executing inside the terminal service - mostly around 25% utilization
- Memory = consistently around 48% to 50% throughout duration of test
- Responsiveness = 100% consistently
- `PerfRun.exe` = 7.2k of Memory

Executing a 5 VUser increasing workload test over a 10 minute period, with VUser per process set to 5

- CPU% = anywhere between 1% to 51% depending on which state the Silk Test process was in while executing inside the terminal service - mostly around 25% utilization
- Memory = constantly around 46% to 48% throughout duration of test
- Responsiveness = 100% constantly
- `PerfRun.exe` (single) = 10.5 k consistently throughout the test

Executing a 10 VUser increasing workload test over a 10 minute period, with VUser per process set to Automatic Calculation (resulting in 10 `PerfRun.exe` files being launched)

- CPU% = anywhere between 1% to 63% depending on which state the Silk Test process was in while executing inside the terminal service, mostly around 23% or 41% utilization
- Memory = constantly around 62% throughout duration of test
- Responsiveness = 100% constantly
- `PerfRun.exe` (each) = 7.5 k consistently throughout the test

Executing a 20 VUser increasing workload test over a 10 minute period, with VUser per process set to 20

- CPU% = anywhere between 17% to 45% depending on which state the Silk Test process was in while executing inside the terminal service, mostly around 25% or 38% utilization
- Memory = constantly around 76% throughout duration of test
- Responsiveness = 100% constantly
- `PerfRun.exe` (each) = 15 k consistently throughout the test

Executing a 25 VUser increasing workload test over a 15 minute period, with VUser per process set to 25

- CPU% = anywhere between 22% to 45% depending on which state the Silk Test process was in while executing inside the terminal service, mostly around 25% or 38% utilization
- Memory = constantly around 83% throughout duration of test (when all VUsers were running)
- Responsiveness = 100% constantly
- `PerfRun.exe` (one) = 15 k consistently throughout the test

Executing a 36 VUser increasing workload test over a 18 minute period, with VUser per process set to 18 (for example, two `PerfRun.exe` files executing all 36 VUsers)

- CPU% = anywhere between 22% to 45% depending on which state the Silk Test process was in while executing inside the terminal service, mostly around 25% or 38% utilization
- Memory = constantly around 98% to 100% throughout duration of test (when all VUsers were running)
- Responsiveness = 100% constantly
- `PerfRun.exe` (each) = 19 k consistently throughout the test

Note: The above values were calculated with no logging enabled (no TrueLog, no `vuser.log`, etc., files being generated) as this would add to the resource consumption of the load test . To summarize, it can be said that both CPU and memory are the limiting scalability factors when executing a GUI-level load test . However, if you specify the setting `VUser per process` to as high a value as possible (although never higher than 25 VUsers per process) then this will significantly reduce both the CPU and the memory utilization of the load test . It is also worth noting that the greater the memory footprint of the application under test, then the less VUsers you will be able to scale successfully during the test.

Additional Information
Additional information on executing the load test on machines which use Multiple Processors:

As long as you have more replay engines (`PerfRun.exe`) running than you have CPUs available on the machine, the current implementation of GUI-level support in Silk Performer does not have any limitations regarding the number of processors that can be used on a Silk Performer agent installation. During a load test scenario, each Silk Performer replay engine (`PerfRun.exe`) will be bound to a particular processor in a round robin method, for example: `PerfRun.exe1` to CPU1, `PerfRun.exe2` to CPU2, `PerfRun.exe3` to CPU3, `perfRun.exe4` to CPU4, and so on.

Troubleshooting GUI-Level Testing Issues

When troubleshooting GUI-Level issues it is important to note that there are three separate components (Silk Performer, Silk Test, and Windows/Terminal Services/Remote Desktop Services) that play integrated roles during the execution of GUI-Level tests; each of these components should be considered when attempting to isolate the root causes of errors.

Step 1: Windows test-environment configuration

Note: For resolutions to issues outlined in this section, please visit the *Micro Focus Knowledge Base* and enter the referenced **Resolution ID**.

The first thing to consider is that Silk Performer can only execute multiple GUI-level virtual users on Microsoft Windows operating systems that have Terminal Services/Remote Desktop Services installed, licensed and configured. If you attempt to execute more than one GUI-level virtual user from a Microsoft Windows machine you will encounter the following error message: `StInitSession(GUI-Level Testing Replay: 10 - Virtual user information, Silk Test Connection timeout reached.`

Resolution ID: `17256`, `17231`

The next, and perhaps most important, step is to configure Windows Terminal Services/Remote Desktop Services to allow each Silk Performer virtual user to execute a Silk Test test case within a separate terminal session. Therefore it is of the vital importance that each of the settings below be configured exactly as specified in the resolution listed below.

Resolution ID: `17255`

Please note that failure to configure Windows Terminal Services/Remote Desktop Services as recommended above can result in error messages such as `GUI-Level Testing Replay: 10 - Virtual user information, RDP not connected`.

Resolution ID: `20117`

Once you have configured Terminal Services/Remote Desktop Services, the final configuration check is to ensure that you are using the correct version of Silk Test (for test case generation) and that you have the correct Silk Performer licenses available for a GUI-level load testing.

Resolution ID: `17168`, `17148`

Step 2: Proxy Server Configuration

In some situations, when recording a Silk Performer script via the Silk Test interface, the resulting BDF file contains no Silk Performer functions. To resolve this issue, perform the following:

1. Launch Internet Explorer and navigate to **Tools** > **Internet Options**.
2. Select the **Connections** tab.
3. Click **LAN settings**. The **Local Area Network (LAN) Settings** dialog box opens.
4. Check the **Use a proxy server for your LAN** check box.
5. In the **Address** field, type `localhost`.
6. In the **Port** field, type `8080`.
7. Click **OK**.

Step 3: Silk Test configuration and test-case generation

Note: For resolutions to issues outlined in this section, please visit the *Micro Focus Knowledge Base* and enter the referenced **Resolution ID**.

When using Silk Test to generate a test case for execution in Silk Performer it is important that you consider that the test case will eventually be executed by Silk Performer within a Terminal Services/Remote Desktop Services/Remote Desktop Services environment. This means that certain considerations need to be made, such as ensuring that a full version of Silk Test is installed on the Silk Performer Agent otherwise Silk Performer will report the error message `GUI-Level Testing Replay: 7 - Application could not be launched`.

Resolution ID: `17181`

Ensure that any directory paths that have been configured for Silk Test are still available when the Silk Test project is exported to Silk Performer; otherwise the Silk Performer runtime engine may be unable to can locate the directory path used to launch the application under test. Failure to set a global path can result in an error message such as `Error: Directory XXXX does not exist"`.

Resolution ID: `17204`

Finally, before exporting the Silk Test project to Silk Performer it is imperative that you export the project using the correct settings. Otherwise you may see the following error: `GUI-Level Testing Replay:11 SilkTest reported. Project failed to open`. The resolution below describes both the consequences of not doing this and the correct way to export a project from Silk Test.

Resolution ID: `17200`

Step 4: Silk Performer configuration and common GUI-level replay errors

Note: For resolutions to issues outlined in this section, please visit the *Micro Focus Knowledge Base* and enter the referenced **Resolution ID**.

The final component to look at when troubleshooting GUI-level issues is Silk Performer. The first thing an end user should consider before they replay a GUI-level BDF script in Silk Performer is that there are major differences between executing a Silk Test testcase within Silk Performer using a normal *console session* and executing a Silk Test test case using a *terminal server session*. The major differences between running a BDF script as a console session and terminal server session are detailed in the following resolution.

Resolution ID: `17258`

Failure to understand the differences between the types of sessions that can be executed in Silk Performer and failure to instruct Silk Performer that you wish to execute a terminal server session can lead to the common replay error `GUI-Level Testing Replay: 10 - Virtual user information, More than 1 user per Session is not allowed`. Refer to the resolution listed below to learn how to avoid this error during replay in Silk Performer.

Resolution ID: `17257`

Other errors that commonly occur during replay are related to the Terminal Services/Remote Desktop Services session in which the Silk Test test case runs. For example is it important to consider that when a Silk Test test case is initially recorded it is often within an operating system environment that uses different user credentials than the environments in which the Silk Test test case will be executed within the terminal server environment. This can result in unexpected windows being generated during replay within the terminal server session and as a result the Silk Test agent will report an error message during replay within Silk Performer such as `Log Error: *** Error: Window 'window name' was not found`. The following resolution provides a good example of one such error and explains how you can avoid it.

Resolution ID: `17236`

Before you execute an actual GUI-level test it is important to consider that there are limitations in regards to the number of virtual users that can be executed within a Terminal Server environment. The resolution below outlines the typical number of GUI-level virtual users that can be executed from a single Silk Performer installation.

Resolution ID: `17202`

Jacada Support

This section explains Silk Performer support for the testing of Jacada applications, including project setup and Java configuration issues.

Jacada is a J2EE-based solution that covers both the agent desktop and the back end. Jacada is mainly used to wrap applications and present a Web UI to clients.

Creating Jacada Projects

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. In the **Type** menu tree, select **Application Server/Component Models** > **Jacada**.
5. Click **Next** to create a project based on your settings.

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

Configuring Jacada Recording Profiles

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. In the **Application** list, select the listed recording profile and click **Edit**. The **Recording Profile** dialog box opens.
4. In the **Protocol selection** area, check the **Web** check box, if it is not already selected.
5. Click **Web Settings**. The **Web Settings** dialog box opens.
6. Click the **WinSock** option button, then click **OK**.
7. In the **Java API** list box, check the **Jacada** check box.
8. Click **OK** to close the **Recording Profile** dialog box.
9. On the **System Settings** dialog box, click the **Proxies** tab to edit the recording proxy configuration. The Proxies page opens.
10. In the **Proxies** list box, select the `SOCKS` protocol and click **Edit**. The **Proxy Settings** dialog box opens.
11. In the **Suppress recording (only forward data)** area, type `1100-1200` in the **Within port range** text box to suppress the recording of forward data.
12. Click **OK** to close the **Proxy Settings** dialog box.
13. Click **OK** to close the **System Settings** dialog box.

Configuring Java Profile Settings for Jacada

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click **Java**.
4. From the Jacada server on which you want to test your Jacada application, copy the file `clbase.jar` to your local working directory.

   The default location of `clbase.jar` on the Jacada server is `\classes\cst\`.
5. Click **File** to locate `clbase.jar` on your local machine and add it to the **Classpath**.
6. Click **OK** to save your profile settings.

Recording Jacada Applications

Record your Jacada application like a standard Web application.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **URL** text box, enter the URL of the Jacada application that you want to test.
   Example: `http://my.jacada.server:9999`
3. Click **Start recording** to begin recording the session.
4. Perform the required interactions with the application under test.
5. When recording is complete, close the application, close the recorder, and save the test script.
6. To confirm that your test script runs as expected, execute a Try Script run in animated mode; you will then be able to view the results of the test run in TrueLog Explorer.

Structure of a Jacada BDL Script

When recording a Jacada application with Silk Performer, the resulting BDL script contains a mix of both Web/HTTP as well as Jacada protocol entries. The script is structured into HTTP traffic and user interactions:

- HTTP traffic
  - `JacCreateUser(…)`
  - `JacConnectUsing…(…)`
- User Interactions
  - `JacDestroy(…)`

Jacada Functions

A number of functions are provided to handle Jacada function calls from within Silk Performer. These functions are one-to-one wrappers of Jacada's own API. Refer to Jacada documentation for detailed information.

Following is a sample of Jacada user interactions:

- Enter Data
  - `JacSetTextData(...)`
  - `JacSetTableData(...)`
- Screen Transition
  - `JacSendWindowData(...)`
  - `JacSendCommand(...)`
  - `JacChangeCellValue(...)`
- Verifications
  - `JacCheckScreen(...)`
  - `JacCheckFieldValue(...)`
  - `JacCheckTableCell(...)`

Java Support

Silk Performer's Java support includes the testing of Enterprise JavaBeans (EJB), RMI objects, JUnit, and more.

Java Framework Support

As a powerful extension to Silk Performer's Benchmark Description Language, the Java Framework enables you to implement user behavior in Java. When testing an existing Java application you do not need to spend much time creating test scripts. The only effort required is embedding existing Java source code into the framework. Refer to the Java Framework Developer Guide for detailed information.

To generate a benchmark executable and run a test with the help of the Java framework, two source files are typically required:

- Java class that implements the behavior of a virtual user
- Silk Performer test script that invokes Java method calls

Both source files need to have the same name, with extensions indicating the file types (`.bdf` for the Silk Performer test script and `.java` for the Java source file).

With Silk Performer it is possible to call Java functions on any Java object from a BDL context with any number and type of parameter and return parameter. When a Java function returns a complex object, a handle for the Java object is returned in its place. The handle can be passed to other function calls as an input parameter, or public methods may be invoked on the object. There is also support for static methods, cast operations, and exception handling.

Creating Java Projects

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 **Java/Java Framework**.
5. Click **Next** to create a project based on your settings.

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

If you need to add additional resources to the project, right-click the project icon in the menu tree. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for the project.

Generating Java Framework Files

1. Click **File** > **New** > **Java Framework Scripts (.bdf/.java)** . The **New Java Framework Script** dialog box opens.
2. Type a name for your Java class.
3. Click **OK**. A Java file containing a class with the specified name is generated and opened along with a BDL file, which is used to test the Java class.
4. If you save your files, you are asked whether you want to add them to your project.

Generating Java Files Without a BDL File

1. Select **File** > **New...** > **Java Script (.java)** from the main menu bar. The **New Java Script** dialog appears.
2. Enter a name for your Java class.
3. Click **OK**. A Java file containing a class with the given name is generated and opened.
4. When saving a new Java file for the first time, you are asked if you want to add the file to the current project. Click **Yes**.

   There are two other ways you can add Java files to your project:

   - Right-click the **Scripts** subnode of the project and select **Add Existing Script...** to open the **Select Script(s)** dialog. Change the **File of type** list box selection to `Java files (*.java)`. Select your file, which will be added to the project when you click **Open**.
   - Right-click in Java file view to open the context menu. Click **Add to Project** to add the file to the project.

Compiling Java Files

1. Select **Settings** > **Active Profile** .
2. Select the **Java** icon in the left-hand scroll window. The **General** page appears.
3. Click the folder icon next to the **Java home** text box to select the directory where the Java SDK is installed.

4. Add any required entries to the classpath in the **Classpath** text box.

   Note: Alternatively, you can use the `JavaSetOption` function with the appropriate option number to set the Java version, home directory, and classpath.
5. Click **OK**.
6. Perform manual compilation using one of the following methods:
   - Open a Java file and press `F7`.
   - Open a Java file and right-click in the view to open the context menu. Select the **Compile** command.
   - Right-click a Java file on the **Project** page to open the context menu. Select the **Compile** command.

Initiating Try Scripts, finding baselines, and running tests will also automatically launch compilation for all Java and BDL files.

During compilation, all output from the Java Compiler is redirected to the **Compiler** page. Resulting class files are generated in the current project directory.

Switching between 32-bit and 64-bit Java

1. In the menu, click **Settings** > **Active Profile**.
2. Click the **Java** icon and then the **Advanced** tab.
3. Select **32-bit Java** (default) or **64-bit Java**.
4. If you use **64-bit Java**, specify an **Execution timeout** for the communication between the Silk Performer runtime and the JVM.

The specified Java architecture determines whether your Java test code is running in-process or out of process. This results in the following behavior:

- **32-bit Java** is enabled: When you execute a test, the `jvm.dll` and the required .jar files are loaded dynamically into the `perfrun.exe`. Note that this increases the memory usage of the `perfrun.exe`. Also be aware that during the early phase of a load test, the memory usage might be volatile. This can be misunderstood as a memory leak, but is in fact expected due to the Java garbage collector at work. By default, up to 50 virtual users share the same JVM, which helps reduce memory usage. However, this feature requires that your Java test code is thread-save (especially the static variables).
- **64-bit Java** is enabled: When you execute a test, the Java test code is running in a separate process - it is not loaded into the `perfrun.exe`.

You can also use the BDL function `JavaSetOption` to switch between 32-bit and 64-bit Java. Note that the settings defined in the BDL script override the options defined in the profile settings.

Testing SOAP Web Services for Java

A Web service is an available service on the Web that can be invoked and from which results can be returned. Although other standards exist, the widely accepted standard for Web services, which has been adopted by the W3C, is SOAP (Simple Object Access Protocol).

This chapter explains the basics of SOAP-based Web Services and details how they can be tested.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol that is used for the exchange of information in decentralized, distributed application environments. You can transmit SOAP messages in any way that the applications require, as long as both the client and the server use the same method. The current specification describes only a single transport protocol binding, which is HTTP.

SOAP perfectly fits into the world of Internet applications and promises to improve Internet inter-operability for application services in the future. In essence, SOAP packages method calls into XML strings and delivers them to component instances through HTTP.

SOAP XML documents are structured around root elements, child elements with values, and other specifications. First an XML document containing a request (a method to be invoked and the parameters) is sent out. The server responds with a corresponding XML document that contains the results.

SOAP is not based on Microsoft technology. It is an open standard drafted by UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft, and SAP. SOAP 1.1 was presented to the W3C in May 2000 as an official Internet standard. Microsoft is one of the greatest advocates of SOAP and has incorporated SOAP as a standard interface in the .NET architecture.

A SOAP stack, an implementation of the SOAP standard on the client side, is comprised of libraries and classes that offer helper functions. A significant Web service testing challenge is that there are a number of SOAP stack implementations that are not compatible with one another. So although SOAP is intended to be both platform- and technology-independent, it is not. Web services written in .NET are however always compatible with .NET clients—they use the same SOAP stack, or library. When testing a .NET Web service however, you need to confirm if the service is compatible with other SOAP stack implementations, for example Java SOAP stack, to avoid interoperability issues.

SOAP client requests are encapsulated within HTTP POST or M-POST packages. The following example is taken from the Internet draft-specification.

Sample Call

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The first four lines of code are standard HTTP. POST is the HTTP verb which is required for all HTTP messages. The `Content-Type` and `Content-Length` fields are required for all HTTP messages that contain payloads. The content-type `text/xml` indicates that the payload is an XML message to the server or a firewall capable of scanning application headers.

The additional HTTP header `SOAPAction` is mandatory for HTTP based SOAP messages, and you can use it to indicate the intent of a SOAP HTTP request. The value is a URI that identifies the intent. The content of a `SOAPAction` header field can be used by servers, for example firewalls, to appropriately filter SOAP request messages in HTTP. An empty string ("") as the header-field value indicates that the intent of the SOAP message is provided by the HTTP Request-URI. No value means that there is no indication on the intent of the message.

The XML code is straightforward. The elements `Envelope` and `Body` offer a generic payload-packaging mechanism. The element `GetLastTradePrice` contains an element called `symbol`, which contains a stock-ticker symbol. The purpose of this request is to get the last trading price of a specific stock, in this case Disney (DIS).

The program that sends this message only needs to understand how to frame a request in a SOAP-complient XML message and how to send it through HTTP. In the following example, the program knows how to format a request for a stock price. The HTTP server that receives the message knows that it is a SOAP message because it recognizes the HTTP header `SOAPAction`. The server then processes the message.

SOAP defines two types of messages, *calls* and *responses*, to allow clients to request remote procedures and to allow servers to respond to such a request. The previous example is an example of a call. The following example comes as a response in answer to the call.

Sample Response

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The first three lines of code are standard HTTP. The first line indicates a response code to the previous POST request, the second and third line indicate the content type and the fourth line the lenght of the response.

XML headers enclose the actual SOAP payloads. The XML element `GetLastTradePriceResponse` contains a response to the request for a trading price. The child element is `Price`, which indicates the value that is returned to the request.

Java Explorer

Java Explorer allows users to create test cases using point and click operations. Java Explorer provides support for the following technologies:

- SOAP Web Services
- RMI
- EJB
- General GUI-less Java objects

Java Explorer can be used to export complete Silk Performer projects that make use of the Silk Performer Java Framework. Java Explorer itself can only run previously defined test scenarios in animated mode. With exported projects however, Silk Performer can perform real tests with multiple virtual users.

Note: Java Explorer currently uses the Axis SOAP stack (*http://www.apache.org*) to generate Web Service client proxy classes.

Java Framework

With the Java Framework, Silk Performer offers a powerful means of simulating virtual users whose behavior is defined with the Java programming language. Arbitrary Java classes can be instantiated within the framework and methods defined within classes may be invoked.

The behavior of virtual users running in the Java framework can be scripted manually using Silk Performer or another Java IDE. A more convenient method however is to use Java Explorer to define virtual user behavior through its point and click interface.

Testing SOAP Over HTTP-Based Web Services

Silk Performer offers three options for testing SOAP over HTTP based services:

- Recording/replaying HTTP traffic
- .NET Explorer in combination with Silk Performer .Net Framework
- Java Explorer in combination with Silk Performer Java Framework

Your environment and prerequisites will determine which of these options is best for your needs.

Recording and Replaying HTTP Traffic

Recording the SOAP protocol over HTTP is as straightforward as recording any Web application that runs in a browser. The application that you record is the application that executes the SOAP Web Service calls. This can either be a client application or a part of the Web application itself.

Creating a New XML/SOAP Project

When you want to record and replay HTTP traffic to test SOAP over HTTP-based Web services, you first need to create a new Silk Performer project of the **Web Services** > **XML/SOAP** type.

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 Services** > **XML/SOAP**. This application type automatically configures its profile settings so that `SOAPAction` HTTP-headers, that are used by SOAP-based applications when calling Web services, are to be recovered.
5. Click **Next** to create a project based on your settings.

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

Creating the Recording Profile

When you want to record and replay HTTP traffic to test SOAP over HTTP-based Web services, you need to create a recording profile for the client application that you want to record.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Click **Settings**. The **System Settings - Recorder** dialog box appears.
3. In the **System** group box, click **Recorder**. The **Recording Profiles** page opens.
4. Click **Add** to add a new recording profile to the list. The **Recording Profile** dialog box opens.
5. Type a name for the recording profile in the **Profile name** text box.

   For example `Internet Explorer`.
6. Click **Browse ...** next to the **Application** path text box and select the path to the application executable.

   For example `C:\Program Files\Internet Explorer\Explorer.exe`.
7. Define the **Working directory**.

   For example `C:\Program Files\Internet Explorer`.
8. Define the **Program arguments**.

   For example `about:blank`.
9. Select the application type from the **Application Type** list box.

   For example `MS Internet Explorer`.

10. In the Protocol selection area, check the check box that corresponds to the protocol that you want to use. For example, check the **Web** check box.
11. To configure the recording profile for WinSock recording click **Web Settings**, which enables you to select the method that the Recorder is to use to capture Web and TCP/IP-based traffic.
12. Click **OK**.

Recording a Script

Record a script with your created recording profile. Interact with your client application and the recorder will record all SOAP requests that are executed over HTTP/HTTPS. When you are finished, close the application and save the recorded script.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select one of the listed browsers from the **Recording Profile** list, depending on the browser you want to use for recording.
3. In the **URL** field, enter the URL that is to be recorded.
4. Click **Start recording**. The Silk Performer Recorder dialog opens in minimized form, and the client application starts.
5. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.
6. Interact with your client application. The recorder records all SOAP requests that are executed over HTTP/HTTPS.
7. To end recording, click the **Stop Recording** button.
8. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.

Script Customization

Each SOAP request that is recorded includes a `WebHeaderAdd` and a `WebUrlPostBin` API call.

You can either customize the input parameter of each Web Service call by manually changing the script or you can use the more convenient method of performing customizations within TrueLog Explorer. To do this, run a Try Script. Then use the XML control to customize the XML nodes that represent the input parameters.

Sample SOAP Request

```
WebHeaderAdd("SOAPAction", "\"http://tempuri.org/Login\"");
WebUrlPostBin(
  "http://localhost/MessageWebService/MessageService.asmx",
  "<?xml version=\"1.0\" encoding=\"utf-8\"?>"
  "<soap:Envelope xmlns:soap=\"http://schemas.xmlsoap.org/soap/
envelope/
\""
    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\""
    "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">"
    "<soap:Body>"
      "<Login xmlns=\"http://tempuri.org/\">"
        "<sUsername>myuser</sUsername>"
        "<sPassword>mypass</sPassword>"
      "</Login>"
    "</soap:Body>"
  "</soap:Envelope>", STRING_COMPLETE, "text/xml;
charset=utf-8");
```

Replaying a Script

Once you have finished script customization, you can replay your script, either in another Try Script run, as part of baseline identification, or in a test.

Select how you want to replay your script. The following options are available:

- Start a Try Script run.
- Replay the script as part of a baseline identification.
- Replay the script in a test.

As the Web service calls are performed along with Web API functions, you receive the same measures you receive when testing any Web application, including detailed protocol-specific statistics.

External References

1. Session, Roger

SOAP. An overview of the Simple Object Access Protocol, March 2000

http://www.w3.org/TR/SOAP/

2. W3C

Simple Object Access Protocol (SOAP) 1.1, December 2000

http://www.w3.org/TR/SOAP/

3. UN/CEFANT, OASIS

Enabling Electronic Business with ebXML, December 2000

http://www.ebxml.org/white_papers/whitepaper.htm

4. Geyer, Carol

ebXML Integrates SOAP Into Messaging Services Specification, March 2001

http://www.ebxml.org/white_papers/whitepaper.htm

5. Open Financial Exchange

Open Financial Exchange Specification 2.0, April 2000

http://www.ofx.net/

Java Over HTTP Support

This section explains Silk Performer support for the testing of applications that rely on Java over HTTP.

Java Over HTTP Overview

Some applications make use of Java Object Serialization to transfer objects between client and server over the HTTP protocol. This communication, based on the exchange of serialized Java objects, uses data in binary format. Therefore Java Object Serialization is effectively object representation in binary format.

JAVA Object Serialization has been used by many applications for many years. For full details, see the *Java specification for object serialization*.

A popular framework that enables the use of Java over HTTP as a remoting technology is Spring Framework and its HTTP Invoker.

Java Over HTTP Project Setup and Testing

Prerequisites

For testing Java over HTTP applications, Java Development Kit 1.6 or later is required.

Transformation is enabled for HTTP requests and responses that have the HTTP header `Content-Type` set to `"application/octet-stream"` or `"application/x-java-serialized-object"`. If you need to transform data with a different HTTP Content-Type header, see *Transformation of Custom Content-Types*.

Setting Up a Java Over HTTP Project

When outlining your project, be sure to select the `Java over HTTP` application type (`Web browser/ Java over HTTP` on the **Outline Project** dialog).

By selecting the `Java over HTTP` application type, several profile settings are automatically configured for you on the tab **Profile** > **Web Protocol Level** > **Transformation**:

- The `Java over HTTP` transformation DLL is selected in the **Type** drop list.
- `Transform HTTP Requests` is enabled
- `Transform HTTP Responses` is enabled
- `Enable Java Virtual Machine usage` is enabled. This setting is required because Java over HTTP requires a running JVM for accurate transformation of serialized/externalizable Java objects. This setting also causes transformed binary Java traffic to appear in readable XML representation.

Modeling a Script

Using the JVM requires the configuration of Java over HTTP application-specific custom JAR files that contain the classes that are necessary for deserialization and transformation into correct XML representation. For this reason, when modeling a Java over HTTP script, the **Model Script** dialog includes an **Add Custom JAR File(s)** button. Click this button to browse to and **Add** any custom JAR files that are specific to the application under test. Added JAR files are displayed within the Project tree **Data Files** node.

Note: If your application is based on Spring Framework HTTP Invoker, you must add `spring.jar` to the classpath.

These required JAR files (or individual `.class` files) are located on the server to be tested. These files must be prepared manually and copied from the server to the Silk Performer controller machine.

Note: It is recommended that JAR files be placed in the `Project` folder.

Clicking **Settings** on the **Model Script** dialog links you directly to the current user profile, **Java Settings** tab. Use this to change Java settings for the currently selected user profile.

Note: Do not enable **Use system classpath** on the **Java Settings** tab. JAR files set in the system classpath may overrule manually configured JAR files.

Generating a Script from a Capture File

While recording a user transaction, the Silk Performer recorder creates a so-called Silk Performer capture file, which contains the entire traffic of the recorded session. After saving, the capture file is opened in the Workbench for further analysis and processing. Before generating a script from the captured traffic, you can configure recording rules and other settings, which are applied during the script generation process.

If any errors occur, click the buttons in the **Resolve Problems** area to resolve them. Then, click **Generate Script** to generate a script from the capture file.

On 64-bit operating systems, both a 32-bit and 64-bit Java installation are required. The 32-bit installation is used for replaying scripts, the 64-bit installation is used for generating scripts. If a 64-bit installation is not available, you can force Silk Performer to use a 32-bit process for script generation by setting the following registry key to `1`: `HKEY_LOCAL_MACHINE\SOFTWARE\Silk\SilkPerformer\<version>\Force32BitCaptureAnalyzer`.

Attention: Using the 32-bit script generator can cause issues with large capture files.

Replaying a Script

If your Java configurations are incorrect (for example, if JAR files are missing), XML responses (only visible in TrueLog Explorer) will not be generated in an easily readable format. Also warnings or errors will be written to the Virtual User Output pane. Typically, errors and warnings indicate whether or not they were caused by requests (client to server) or responses (server to client).

Customizing Java Over HTTP Scripts

Do not change the overall structure of XML objects within Java over HTTP scripts. It is okay to parse values or insert verification functions, but deleting or rearranging Java over HTTP XML elements will destroy a call. Also do not change the order of elements within arrays.

Customizing Java Over HTTP Scripts

You can use either Silk Performer or TrueLog Explorer (**Response** tab) to customize Java over HTTP scripts.

Customizing Input Data Using Silk Performer

1. Search for the original value in the script, this is typically an XML node value.
2. Select the string you want to customize.
3. Right click and chose **Customize Selected String**.
4. Follow the steps outlined by the string customization wizard.

Note: Do not change the overall structure of XML objects within Java over HTTP scripts. It is okay to customize input, parse values and insert verification functions, but deleting or rearranging Java over HTTP XML elements can destroy a call. Also, do not change the order of elements within arrays.

Customizing Input Data Using TrueLog Explorer

1. Run a Try Script.
2. Open the resulting TrueLog (XLG) file.

   To do this, from within TrueLog Explorer, click **Open TrueLog**, open the related project folder, and double click the XLG file.
3. Expand the TrueLog node in the tree menu and select a request.
4. Select the **Request** tab.
5. Select the XML node value you want to customize.

   Tip: Right-click in the rendered XML view and choose **Find** to search for an XML node value. The global find option is not able to search rendered XML view.
6. Right click and choose **Customize Selected String**.
7. Follow the steps outlined by the string customization wizard.

Adding Verification and Parsing Statements

This section explains how to use TrueLog Explorer to add verification statements using both rendered XML view and plain-text source view.

Adding Verification Statements Using Rendered XML View

1. Run a Try Script.
2. Open the resulting TrueLog (XLG) file.

   To do this, from within TrueLog Explorer, click **Open TrueLog**, open the related project folder, and double click the XLG file.

3. Expand the TrueLog node in the tree menu and select a request.
4. Select the **Response** tab.

   Note: XML can only be rendered when an appropriate content type is sent from the server. Use plain-text source view for unsupported content types.
5. *(Optional)* If you are sure that an unknown application type (for example, octet-stream) can be rendered as XML, you may do the following to render it:
   a) Run `regedit`.
   b) Select `HKEY_CURRENT_USER\Software\Silk\TrueLog Explorer\...\TreatAsXmlAlways`
   c) Add `application/octet-stream` to the list of values. Use a semi-colon (`;`) as a separator.
   d) Restart TrueLog Explorer.
6. Right-click an XML node value and choose **Verify Element Value…** or **Parse Element Value…**.

   Tip: Right-click in the rendered XML view and choose **Find** to search for an XML node value. The global find option is not able to search rendered XML view.
7. Follow the steps outlined by the wizard.

Adding Verification Statements Using Plain-Text View

1. Run a Try Script.
2. Open the resulting TrueLog (XLG) file.

   To do this, from within TrueLog Explorer, click **Open TrueLog**, open the related project folder, and double click the XLG file.
3. Expand the TrueLog node in the tree menu and select a request.
4. Select the **Response** tab.
5. Search for and select the value to be verified.
6. Right-click an XML node value and choose **Verify Element Value…** or **Parse Element Value…**.
7. Follow the steps outlined by the wizard.

JUnit Integration

Silk Performer facilitates smooth integration of existing JUnit test scripts (version 3.8.x and 4.x) into Silk Performer for the support of remote-component testing under realistic concurrent-access server conditions.

Silk Performer provides an import tool that enables you to import existing JUnit test classes and other Java classes. Tested Java methods may have parameters and return values; code for setting the in-parameters of these functions is generated automatically. By definition, JUnit test methods cannot have parameters.

Silk Performer's Unit Test Import tool offers you the option of selecting specific test methods. It automatically generates BDL stub code (a benchmark description file) that calls those selected test methods. Existing JUnit test classes can be called from Silk Performer without requiring modification of the test classes.

Prerequisites

JUnit 4.x testing requires Java version 1.5 or higher.

The respective JUnit libraries must be available in the classpath of the Java settings of the active profile.

Setting Up a New JUnit Silk Performer 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, expand **Unit Testing** and select **JUnit**.
5. Click **Next**.

   Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.

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

Importing a JUnit or Java Test Class

Modify your Java profile settings accordingly before importing JUnit 4.x module tests.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **File** field, specify the archive that is to be tested. The archive is automatically added to the profile classpath. The available classes are then retrieved and displayed, sorted alphabetically in the **Class** field.
3. From the **Class** list, select one of the available classes for testing.

   When you do not specify a specific archive for testing, the wizard enables you to specify a class that is available in the profile classpath. Type the fully qualified class name into the **Class** field, for example `java.lang.String`.

   The available constructors and methods are automatically retrieved and displayed.
4. From the **Constructor** list, select the appropriate constructor for instantiating the imported class.

   If a constructor is not selected (entry `<No Instance Required – Only Static Calls>`), only static methods will be displayed in the method list.
5. In the **Methods** area, select the methods that you want to call.
6. To filter the methods that are shown in the **Methods** area, perform the following steps:
   a) Click the **Advanced Settings** button (the funnel icon above the **Methods** area).
   b) Once you have customized filter settings, click **OK** to update the **Methods** area.
7. To change general Java settings including the Java version, Java home directory, or JVM DLL, click the **Active Profile Settings** link. The **Profile Settings** dialog opens to the **Java/General** page for Java projects (JUnit project type).

   Note: Changes made to these settings (for example Java Classpath) may lead to different results. Selections made in the **Class**, **Constructor**, and **Methods** fields will be updated with the new results.

   Note: If you change the Java version, Java home directory, or JVM DLL, you must restart Silk Performer for the changes to take effect.
8. Click **OK**.

Java Profile Settings

The JUnit or Java test class you want to import and all third-party libraries needed by the test class must be available in the profile classpath. Silk Performer automatically adds all archives entered in the **File** field to the profile classpath. Also, all test classes compiled in Silk Performer are compiled to the project directory, which is also automatically included in the profile classpath.

Filter Options for JUnit Methods

This table lists the filters that are available in the **Methods** area when you import a test class.

JUnit Method	Description
Parameterless Functions Only	Ignore all functions that expect parameters
Unit Test Functions Only (selected by default for JUnit import)	Ignore all functions that are not JUnit
Member Functions Only	Ignore all non-member functions
Public Functions (selected by default)	Include public functions
Protected Functions	Include protected functions
Private Functions	Include functions with private access
Package Functions	Include functions with package access
Declared Functions Only	Ignore functions from the base classes
Complex Functions (selected by default)	Show functions that take complex parameters. In Java, complex parameters are scripted by JavaSetObject with NULL as the default value.
Autodetect Unit Test Functions (selected by default)	Automatically detect and script JUnit functions according to the following rules: • JUnit 3.8.x scripts are derived from `junit.framework.TestCase` class. • JUnit 3.8.x functions must not have parameters or return values. • setUp() and tearDown() are invoked prior to and following each JUnit 3.8.x method. • JUnit 4.x scripts no longer require to be derived from `junit.framework.TestCase` class. • JUnit 4.x functions are recognized by the `@Test` annotation and added to the script, unless the `@Ignore` annotation is used. • Methods having `@BeforeClass` and `@AfterClass` annotation are invoked once per test class. • Methods having `@BeforeClass` and `@After` annotation are invoked prior to and following each JUnit 4.x method.

Example BDL Script for JUnit 3.8.x (legacy)

Script Example

The following script is generated by importing a sample JUnit 3.8.x test case and selecting the three methods `testRound`, `testSqrt`, and `testMax`:

Note: The following script is a legacy script, which is generated within older Silk Performer versions. For Silk Performer 19.5 and newer, a JUnit import creates a slightly different script, which is documented in *Example BDL Script for JUnit* .

```
transaction TInit
var
    hPerf : number;
begin
    JavaCreateJavaVM();
    JavaSetString(JAVA_STATIC_METHOD, "TestCaseName");
    ghTestObj := JavaLoadObject("JUTestClass", "JUTestClass.<init>");
end TInit;

transaction TMain
begin
```

```
    JUnitCallMethod(ghTestObj, "testRound", "testRound");
    JUnitCallMethod(ghTestObj, "testSqrt", "testSqrt");
    JUnitCallMethod(ghTestObj, "testMax", "testMax");
end TMain;

transaction TEnd
begin
    JavaFreeObject(ghTestObj);
end TEnd;
```

`JavaLoadObject` in the `TInit` transaction instantiates the JUnit test class, `JUTestClass`.

`JUnitCallMethod` in the `TMain` transaction calls one of the three test methods in the same way that a JUnit test runner would. First the `setUp()` method is invoked, then the test method itself (e.g., `testSqrt()`), finally the `tearDown()` method is invoked.

Timers

When an optional timer parameter is specified for a Java method call, the execution times of the constructor, test method, setup method, and teardown method are measured. For the example in this topic, you receive the following measures:

- For the constructor: `JUTestClass.<init>`
- For the methods: `testRound`, `testSqrt`, `testMax`
- For the set up methods: `testRound_setup`, `testSqrt_setup`, `testMax_setup`
- For the tear down methods: `testRound_tearDown`, `testSqrt_tearDown`, `testMax_tearDown`

Example BDL Script for JUnit 4.x (legacy)

Script Example

The following sample JUnit test script contains several annotations for test setup and teardown and is generated by importing a sample JUnit 4.x test case and selecting the methods `doFoo` and `doFoo2`.

Note: The following script is a legacy script, which is generated within older Silk Performer versions. For Silk Performer 19.5 and newer, a JUnit import creates a slightly different script, which is documented in *Example BDL Script for JUnit* .

```
var
  ghTestObj :number;
dcluser
  user
    JavaUser
  transactions
    TInit :begin;
    TMain :1;
    TEnd :end;

dclfunc
  function JUnit4Before
  begin
    // @Before
    JavaCallMethod(ghTestObj, "myBefore", "myBefore");
  end JUnit4Before;

  function JUnit4After
  begin
    // @After
    JavaCallMethod(ghTestObj, "myAfter", "myAfter");
  end JUnit4After;

  function JUnit4CallFunc(hJavaObj :number; sName :string; sTimerName :string
optional; sExceptionBuffer :string optional) : boolean
  begin
    JUnit4Before();
    // @Test
    JUnit4CallFunc := JavaCallMethod(hJavaObj, sName, sTimerName);
    if not StrIsNull(sExceptionBuffer) then
      JavaGetLastException(hJavaObj, sExceptionBuffer);
    end;
    JUnit4After();
  end JUnit4CallFunc;

dcltrans
  transaction TInit
  var
    hPerf : number;
  begin
    JavaCreateJavaVM();
    ghTestObj := JavaLoadObject("JU4ImporterTest", "JU4ImporterTest.<init>");
    // @BeforeClass
    JavaCallMethod(JAVA_STATIC_METHOD, "JU4ImporterTest.myBeforeClass",
"myBeforeClass");
  end TInit;

  transaction TMain
  var
    sBuffer :string;
  begin
    JUnit4CallFunc(ghTestObj, "doFoo", "doFoo");
    JUnit4CallFunc(ghTestObj, "doFoo2", "doFoo2");
  end TMain;

  transaction TEnd
  var
    sBuffer :string;
  begin
    // @AfterClass
    JavaCallMethod(JAVA_STATIC_METHOD, "JU4ImporterTest.myAfterClass",
"myAfterClass");
    JavaFreeObject(ghTestObj);
  end TEnd;
```

In the `dclfunc` section, the helper functions for test setup, teardown, and exception handling are defined. These functions are used in the transactions.

`JavaLoadObject` in the `TInit` transaction instantiates the JUnit test class `JU4ImporterTest`. All JUnit methods that use the `@BeforeClass` annotation are called in the `TInit` transaction.

`JUnit4CallFunc` in the `TMain` transaction calls all the test methods that were selected for the JUnit test import. First, the methods that use the `@Before` annotation are invoked, and then the test method itself, such as `doFoo()`, is invoked. Finally, the methods that use the `@After` annotation are invoked.

Timers

When an optional timer parameter is specified for a Java method call, the execution times of the constructor, test method, setup method, and teardown method are measured. For the example in this topic, you receive the following measures:

- Constructor – `JU4ImporterTest.<init>`
- For the methods – `doFoo`, `doFoo2`
- For the test setup – `myBeforeClass`, `myBefore`
- For the test teardown – `myAfterClass`, `myAfter`

Example BDL Script for JUnit

Script Example

The following sample script is generated by importing a sample JUnit test case.

```
var
  hTestClass : number;

dcltrans
  transaction TInit
  begin
      // Load and start the JVM.
      JavaCreateJavaVM();

      // instantiate the java class
      hTestClass := JUnitLoadClass("com/microfocus/Test");
  end TInit;

  transaction THello
  begin
    JUnitExecuteTest(hTestClass, "doHello", "doHello_timer");
    JUnitExecuteTest(hTestClass, "", "Test_timer");
  end THello;

  transaction TEnd
  begin
    JavaFreeObject(hTestClass);
  end TEnd;
```

- `JUnitLoadClass` in the `TInit` transaction instantiates the Java object and returns a handle of the runtime class. The handle is assigned to `hTestClass`.
- `JavaFreeObject` in the `TEnd` transaction releases the Java runtime class.
- `JUnitExecuteTest` in the `THello` transaction starts the JUnit test execution.

Exception Handling for JUnit 4.x and newer

JUnit 4.x and newer versions allow the definition of methods that throw exceptions. Throwing such exceptions is validated by JUnit. Silk Performer imports JUnit methods that throw exceptions including the validation of the thrown message.

```
//@Test(expected=java.lang.IllegalArgumentException)
    JavaRegisterException(500,"java.lang.IllegalArgumentException",
JAVA_OPTION_MATCH_SUBSTRING);
    ErrorAdd(FACILITY_NATIVE_JAVA, 500, SEVERITY_INFORMATIONAL);
    JUnit4CallFunc(ghTestObj, "doFoo", "doFoo", sBuffer);
    ErrorRemove(FACILITY_NATIVE_JAVA, 500);
    if (StrSearch(sBuffer, "java.lang.IllegalArgumentException",
STR_SEARCH_FIRST) = 0) then
          RepMessage("method did not throw:
java.lang.IllegalArgumentException", SEVERITY_ERROR);
    end;
```

The sample above tests a JUnit method that is expected to throw a `java.lang.IllegalArgumentException`. The function `JavaRegisterException` assigns the BDL error code 500 to the exception. The function `ErrorAdd` sets the error severity to informational because the error is expected. After that, the JUnit method `doFoo` is invoked, where the parameter `sBuffer` retrieves the error message. The `ErrorRemove` function sets the error handling back to its original state. It also verifies that the expected exception occurred, and throws a BDL error if this is not the case.

Note: When modifying the BDL script manually, make sure that you do not map a Java exception to different BDL error numbers, or vice versa.

Importing a JUnit Test Suite

Modify your Java profile settings accordingly before importing JUnit 4.x module tests.

1. Select **Project** > **Import Unit Test** > **JUnit Test Class** . The **Import Unit Test - JUnit Test Class** dialog opens.
2. Cick the **Advanced Settings** button (the funnel icon above the **Methods** area). The **Advanced Settings** dialog opens.
3. Perform the following steps:
   a) Uncheck the **Unit Test Functions Only** check box.
   b) Check the **Declared Functions Only** check box.
   c) Uncheck the **Member Functions Only** check box.
   d) Click **OK**.
4. In the **File** field, specify the archive that is to be tested. The archive is automatically added to the profile classpath. The available classes are then retrieved and displayed, sorted alphabetically in the **Class** field.
5. From the **Class** list, select one of the available classes for testing.

   When you do not specify a specific archive for testing, the wizard enables you to specify a class that is available in the profile classpath. Type the fully qualified class name into the **Class** field, for example `java.lang.String`.

   The available constructors and methods are automatically retrieved and displayed.
6. From the **Constructor** list, select the appropriate constructor for instantiating the imported class.

   If a constructor is not selected (entry `<No Instance Required – Only Static Calls>`), only static methods will be displayed in the method list.
7. In the **Methods** area, select the `suite()` method, then click **OK**.

Executing a JUnit Test Suite

When JUnit Import Tool (**Project** > **Import Unit Test** > **JUnit Test Class**) script code is used to invoke a method that has a return value, the code used to retrieve the return value is scripted, but commented out.

If a Java method returns a `junit.framework.Test` method, additional commented-out code is scripted, which allows the `FRunTestSuite` function from `junit.bdh` to execute the test suite. `junit.bdh` is automatically added to the include section of the BDL script.

Uncomment the code for retrieving the return value.
For example:

```
transaction TMain
begin
  // junit.framework.Test suite()
  JavaCallMethod(JAVA_STATIC_METHOD, "JU4ImporterTestSimple.suite",
"suite");
  // hObj := JavaGetObject(JAVA_STATIC_METHOD);
  // use the following function to execute JUnit test suites
  // FRunTestSuite(hObj);
  // JavaFreeObject(hObj);
end TMain;
```

Java RMI Support

This section explains Silk Performer's support for the record and replay of Java Remote Method Invocation (Java RMI) applications, including project setup and Java configuration issues. It also shows usage of Product Manager, which is the sample Java RMI application that ships with Silk Performer.

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines.

Creating Java RMI Projects

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. In the **Type** menu tree, select **Java** > **Java RMI/EJB (recording)**.
5. Click **Next** to create a project based on your settings.

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

Configuring Java RMI Recording Profiles

1. From the Silk Performer menu bar, choose **Settings** > **Active Profile**.
2. Click **Java** and configure Java version, classpath settings, and classpath elements.

   The recorder uses the JDK internally and the Silk Performer Java Framework uses the JDK for replay.

   In the **Version** list box, select a JDK version that is later than version 1.2.
3. In the Silk Performer menu, click **Settings** > **System** .
4. Click the **Recorder** icon. The **Recording Profiles** page opens.
5. Click **Add** to define a recording profile for the application that you want to record. The **Recording Profile** dialog box opens.
6. Type a name for the new recording profile in the **Profile name** text box.
7. Type the full path to the executable or startup script, that starts your Java RMI client, in the **Application path** text box. Specify the working directory, followed by a path separator. Or browse for the executable, in which case the working directory is inserted automatically.
8. *Optional:* If the executable that hosts the Java Virtual Machine (JVM) is located in another directory than the application, check the **Record executable is different from application path** check box. In the **Record executable** list box, select the executable.
9. In the **Protocol selection** area, check the Java APIs that you want to record in the **Java API** list box.
10. Click **OK** to close the **Recording Profile** dialog box.
11. Click **OK** to close the **System Settings** dialog box.

Configuring Java Profile Settings for the IBM JVM

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. In the **Application** list, select the listed recording profile and click **Edit**. The **Recording Profile** dialog box opens.
4. Click **Java Settings** and select the **Manual** option button.
5. Click **OK** to close the **Java Settings** dialog box.
6. Customize the startup script.
   a) Open the startup-script file, for example `startMyApp.cmd`, in an editor.
   b) Rename the startup-script file, for example to `startMyAppForRecording.cmd`.

c) Insert a call to `perfPrepareJavaRecording.cmd` in the Silk Performer home directory. For example call `C:\Program Files\Silk\Silk Performer 20.0\perfPrepareJavaRecording.cmd`.
d) In the startup-script file, replace `java my.rmi.application <parameters>` with `java %PERFREC_OPTIONS% my.rmi.application <parameters>`, where `my.rmi.application` is an example. Thus you add the command-line parameter `%PERFREC_OPTIONS%` to the java execution.

Recording Java RMI Applications

Record your Java RMI application like a standard Web application.

Important: When you record applets, use a JVM by Sun Microsystems instead of one provided by Microsoft.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. In the **Recording Profile** list box, select the profile for the application that you plan to test, for example `My Rmi Application`. If a profile has not yet been set up for the application you want to use, click **Settings** to the right of the list box to set one up.
3. Click **Start recording** to begin recording the session.
4. Perform the required interactions with the application under test.
5. When recording is complete, close the application, close the recorder, and save the test script.
6. To confirm that your test script runs as expected, execute a Try Script run in animated mode; you will then be able to view the results of the test run in TrueLog Explorer.

Java RMI Recording Results

When you record a Java RMI application, Silk Performer generates two result files, `x.java` and `x.bdf`.

The Java source file `x.java` contains the recorded actions. This file is compiled automatically from within Silk Performer. You have to specify the classpath needed to compile this file in **Settings** > **Active Profile** > **Java**.

The file `x.bdf` contains the BDL stub code that launches the Java virtual user.

Replaying a Java RMI Application

1. Remove the recorded API calls `JavaSetOption(JAVA_CLASSPATH)` and `JavaSetOption(JAVA_CMDLINE_OPTIONS)` from the BDL script.
2. Obtain the JAR and CLASS files from the server. Use the recorded comments in the recorded Java source code to find the JAR and CLASS files.

   The comments look as described in the following example.

```
// ###ClassPath###
// ###Applet CodeBase###
// ###Applet parameters queried by the Applet###
  // codebase=...
  // archives=...
  // code=...
```

3. Copy these files to your local machine.
4. Add the path to these files to the classpaths in **Settings** > **System** > **Java**.

Java RMI Sample Application

The sample application that ships with Silk Performer to demonstrate how to record and replay Java RMI applications is called Product Manager.

To launch Product Manager, navigate to `<public user documents>\Silk Performer 20.0\SampleApps\RMILdap\Web`, and click `ProductManager - DemoVersion.htm`.

Recording and Replaying the Sample Application

1. Create a new Java RMI project.
2. Modify the existing Internet Explorer recording profile. See *Modifying the Internet Explorer Profile*.
3. Specify the script generation details in **Settings** > **Active Profile** > **Record** > **Script** > **Java**.
4. Go to **Settings** > **Active Profile** > **Record** > **Java**, and add the JAR file `ProductManager.jar` from the Silk Performer samples directory, `<public user documents>\Silk Performer 20.0\SampleApps\RMILdap\Web\ProductManager.jar`, to the classpath.
5. Start the RMI and LDAP servers. To start the servers, open the Product Manager sample HTML page, `<public user documents>\Silk Performer 20.0\SampleApps\RMILdap\readme.htm`, and follow the instructions in the *Starting up servers* section.
6. Click **Model Script** in the workflow bar.
7. Type the location of the RMI sample-application HTML page in the **Command line** text box. The default path of the HTML page is `<public user documents>\Silk Performer 20.0\SampleApps\RMILdap\Web\ProductManager - DemoVersion.htm`.
8. Click **OK** to launch the applet. The applet loads and the recorder starts to record the JNDI and RMI activities of the applet. You can now perform searches for products, change products, add new products, and so on.
9. Click **OK** when you have performed all the desired actions. The **Save As** dialog box opens.
10. Specify a valid Java identifier for the script name and click **Save** to save the recorded script.
11. Click **Yes** twice to close the recorder and to confirm the closing.
12. Comment out or delete the recorded API calls `JavaSetOption(JAVA_CLASSPATH, ...` and `JavaSetOption(JAVA_CMDLINE_OPTIONS, ...`.

You can now replay the recorded script. Click **Try Script** on the workflow bar for a try run of the script.

Modifying the Internet Explorer Profile

1. Click **Model Script** on the Silk Performer workflow bar.
2. Click **Settings** next to the **Recording Profile** list box.
3. Select Internet Explorer from the recording profiles.
4. Click **Edit**.
5. Uncheck the **Web** check box.
6. Check the **Java RMI** and **Java JNDI** check boxes.

   This sample uses the Java Naming and Directory Interface (JNDI) for the bootstrapping of remote object references.

Eclipse Plug-In

Silk Performer is packaged with a plug-in for Eclipse, the popular Java IDE that is based on plug-in technology. The plug-in enables Eclipse users to be supported with Silk Performer load testing features.

The structure of the Silk Performer test class supports Java developers in creating test cases for Silk Performer. The test class implements the `ITestClass` interface that is packaged with the Eclipse plug-in. It contains a Silk Performer object that represents the interface to Silk Performer's Java Framework.

The test class includes two basic methods:

- `testInit` - This method initializes the interface. It is called at the beginning of test runs.
- `testEnd` - This method closes the interface. It is called at the end of Try Script runs and load tests.

Between `testInit` and `testEnd` there may be one or more test methods invoked by a running load test. Each method that has a signature `void testXY()` throws `Silk Performer Exception.`

Example:

```
public void testMyMethod() throws
 silk.performer.SilkPerformerException
```

Using the `JavaSetOption` BDL Function

For debugging Java framework projects, some debug parameters must be set as command line arguments. This is done by the Eclipse plug-in automatically when you run a Try Script.

In most cases, running and debugging Try Script tests without modifications is possible. However, when you override the default value of a JVM option in a BDL script using `JavaSetOption`, additional configuration is required to enable debugging in Eclipse.

You must also define an Eclipse debug Remote Java application configuration. After these options are set, you can then launch a Try Script run. Afterwards, you must start the Eclipse debug configuration to start a debugging session.

Eclipse Project Restrictions

There are restrictions on concurrent use of Eclipse and Silk Performer. Projects are locked from use by Eclipse while Silk Performer works on them. If you attempt to open a project from Eclipse that is currently in use by Silk Performer, you will receive an error message. You must close Silk Performer to pass project control back to Eclipse.

Installing the Eclipse Plug-In

You can download the Eclipse Plug-In from the *product updates site*.

The Eclipse SDK and Silk Performer must be installed on your computer. Make sure to run Silk Performer at least once before you start the installation.

1. Use the standard procedure for installing Eclipse plug-ins from the Eclipse SDK (**Help** > **Install New Software**).

   If you are not sure how to install plug-ins in Eclipse, refer to the Eclipse documentation.
2. When asked for the site to add, specify the site that contains your Silk Performer `Extras` folder.

   The `Extras` folder is located in the Silk Performer installation folder.
3. Browse to and select the Eclipse Plug-in directory in your Silk Performer `Extras` directory (`.../Extras/eclipseplugin/`).
4. Make sure that the **Group items by category** check box is not checked.
5. Once the Silk Performer Eclipse Plug-in is listed, click the plus sign of the new local site to expand its contents.
6. Check the check box of the plug-in feature **(Silk Performer Feature)** and then click **Install**. The **Install** dialog opens.
7. Click **Next**. A license description appears on the **Install** dialog.
8. If you agree to the terms of the license agreement, check the **I accept the terms of the license agreement** check box and then click **Finish**.
9. Click **Yes** to restart the Eclipse Workbench. When the Eclipse Workbench is restarted, a Silk Performer menu entry appears in the Eclipse Workbench.

Eclipse Plug-In does not work on Microsoft Windows 8 and Microsoft Windows Server 2012

Problem:

The Silk Performer Eclipse Plug-In does not work on Microsoft Windows 8 and Microsoft Windows Server 2012

.

Resolution:

On Microsoft Windows 8 and Microsoft Windows Server 2012, run Eclipse in compatibility mode. To do this, perform the following steps:

1. Right-click `eclipse.exe` and select **Properties**.
2. On the **Properties** dialog box, select the **Compatibility** tab.
3. In the **Compatibility mode** section, check **Run this program in compatibility mode for:** and select `Windows 7`.

Configuring Eclipse Plug-In Settings

1. In Eclipse, select **Window** > **Preferences** . The **Preferences** dialog opens.
2. Select the Silk Performer node on the menu tree to display Silk Performer settings.
3. Confirm that the installation directory of your Silk Performer installation is listed in the **Install directory** text box. If it is not listed, click **Browse** to browse for and select your Silk Performer installation directory.
4. Edit the **Default project name** as required.
5. Edit the **Default test class name** as required.
6. (experienced users only) Edit the default **Timeout for connecting to Java debugger** setting as required.

   This is the amount of time allowed (in milliseconds) for connecting to a Silk Performer process during tests.
7. (experienced users only) Edit the default **Shutdown after finish delay** setting as required.

   This is the amount of time (in milliseconds) that Eclipse waits before retrieving results after the completion of Silk Performer tests.
8. Select a **Log level** to specify when logs are to be written.

   Selecting **None** results in no log files being written. Selecting **Errors** results in log files being written only when errors are encountered. Selecting **Debug** results in all activity being logged.
9. In the menu tree, expand **Java** and select the **Installed JREs** node. The **Installed JREs** page opens and lists the Java runtimes that are used by Eclipse on your local system.
10. Confirm the **Location** and **Type** settings for your Java runtime environment.
11. To add a Java runtime profile, perform the following steps:
    a) Click the **Add** button. The **Add/Edit JRE** dialog opens.
    b) Click **[...]** to browse to and select the directory that contains your JDK installation (specifically, your Java compiler), then click **OK.** The fields on the **Add/Edit JRE** dialog are automatically populated with values derived from your JDK installation.

       Note: You must select a JDK (version 1.5 or higher), rather than a JRE, because Silk Performer attempts to recompile Java files in its test classes.
    c) Click **Finish** to save your settings.
12. To edit an existing Java runtime profile, perform the following steps:
    a) Click the **Edit** button. The **Add/Edit JRE** dialog opens.
    b) Click **[...]** to browse to and select the directory that contains your JDK installation (specifically, your Java compiler), then click **OK.** The fields on the **Add/Edit JRE** dialog are automatically populated with values derived from your JDK installation.

Note: You must select a JDK (version 1.5 or higher), rather than a JRE, because Silk Performer attempts to recompile Java files in its test classes.

c) Click **Finish** to save your settings.

13. On the **Installed JREs** page, click **OK** to save your settings.

Projects

Setting Up a New Eclipse Project

1. Within a Silk Performer-enabled Eclipse workbench, select **File** > **New** > **Project** . The **New Project wizard** opens.
2. Expand the Silk Performer node in the menu tree.
3. Select the **Java Project** node and then click **Next**. The **New SilkPerformer load testing capable Java project** page opens.
4. Edit the name of your new project as required.
5. Edit the **Test class name** (the name of the initial test class template) as required.
6. Click **Finish**. The plug-in generates the test class template in your Eclipse workspace and links the test classes to a test case in your Silk Performer project.

A class path links the new test class into Silk Performer's workspace so that it works with both the Eclipse editor and Silk Performer. Behind each test class is BDF stub code. This code is generated each time a test class is compiled, so any changes you make to BDF stub code will be overwritten the next time the test class is compiled.

Importing Projects

Another way of linking your Eclipse and Silk Performer workspaces is to import an existing Silk Performer project into your Eclipse workspace.

A Silk Performer Java Framework project that has been created using the Silk Performer Eclipse Plug-In fulfills the required Eclipse notation, used by Eclipse Plug-In projects. A Silk Performer Java Framework project that has been created with Silk Performer or Silk Performer Java Explorer does not fulfill the required Eclipse notation.

When importing a Silk Performer Java Framework project into Eclipse, the result differs as follows:

Project created with Silk Performer Eclipse Plug-In	Project created with Silk Performer or Silk Performer Java Explorer
Running and debugging try script tests without modifications is possible	Limited execution and debugging functionality available
Automatic `bdf` stubfile generation for changes to the test class or test methods	No automatic `bdf` stubfile generation
New test methods can be added to project	No introduction of new test methods possible
Changing existing test class and test method names is possible	Changing existing test class and test method names is not possible

Note: Modifications to Java test code are possible for either imported project type.

Before you can import a Silk Performer Java Framework project that has been created with Silk Performer or Silk Performer Java Explorer into Eclipse, the test class must have the same name as the `bdf` file, for example `TestClass.java` and `TestClass.bdf`.

Tip: Java Explorer allows you to specify the test class and the script file name. Refer to Silk Performer Java Explorer Help for detailed information.

You can customize a Silk Performer Java Framework project that has been created with Silk Performer or Silk Performer Java Explorer manually so that it fulfills the required Eclipse notation used by the Silk Performer Eclipse Plug-In.

Importing a Silk Performer Project into Eclipse

1. Select **File** > **Import** . The **Import** dialog box opens.
2. Expand the **Other** node in the menu tree.
3. Select **Import Existing Silk Performer Java Project**.
4. Click **Next** to view all available Silk Performer projects in your Silk Performer `projects` directory.
5. Check the check boxes of the projects that you want to import.
6. Click **Finish**. The selected project, settings, and classpaths are imported into your Eclipse workspace.

Customizing a Silk Performer Project to Use the Required Eclipse Notation

Before you begin this task, import your Silk Performer project into Eclipse.

You can customize a Silk Performer Java Framework project that has been created with Silk Performer or Silk Performer Java Explorer manually so that it fulfills the required Eclipse notation used by the Silk Performer Eclipse Plug-In.

1. Open the test class in Eclipse and modify it according to the following rules:
   a) The test class must implement the `silk.performer.ITestClass` interface.
   b) The method for test initialization must be named `testInit` and take a single parameter of type Silk Performer.
   c) The method for test finalization must be named `testEnd`.
   d) All test methods must start with the prefix `test` and may not have any parameters.
2. Save your changes.

Configuring Silk Performer Project Properties from Eclipse

1. In Eclipse, select **Project** > **Properties** .
2. Select the **Silk Performer Project Properties** node in the menu tree. At the top of the **Silk Performer Project Properties** dialog box, the path to the targeted Silk Performer project is displayed.
3. If required, edit the **Name** that has been specified for the project when opened in Silk Performer.
4. Edit the **Description** that has been defined for the project, as required.
5. You can define Silk Performer project attributes for the project (Name, value Type, Value, Default value, Description).

   See *Project Attributes* for full details regarding configuring Silk Performer project settings.
6. Click **Apply** to apply your changes.
7. Click **OK**.

   Note: Silk Performer project attributes can also be accessed directly by selecting **Project Attributes** from Eclipse's Silk Performer menu.

Configuring Silk Performer System Settings in Eclipse

1. In Eclipse, select **Silk Performer 20.0** > **System Settings** .
2. Set the configuration options as needed.
3. Click **OK** to save your changes.

Configuring Silk Performer Project Settings in Eclipse

Note: When Eclipse controls a Silk Performer project, all Java settings defined in Silk Performer are overruled by the settings you defined in Eclipse.

1. In Eclipse select **Silk Performer 20.0** > **Profile Settings** . You will be directed to the **General** page for your project profile Java settings, where you can view the Java home and classpaths that have been set up for your newly created project.
2. Set the profile options as needed.
3. Click **OK** to save your changes.

Redirecting Java Build Paths

There are restrictions for classpaths of Eclipse projects that are used within Silk Performer. To enable the distribution of libraries during a test, all libraries must be located either in the Silk Performer project directory (`<my documents>\Silk Performer 20.0\Projects`) or they must be available on a UNC path.

1. Right-click a Silk Performer-enabled project in Eclipse's Package Explorer and select **Build Path** > **Configure Build Path** from the context menu.
2. Click the **Libraries** tab. All of the user libraries that are required for running tests are displayed.
3. Click **Add Library**. The **Add Library** page opens.
4. Select **User Library** from the list and click **Next**.
5. Click **User Libraries**. The **User Libraries** page opens.
6. Click **New**. The **New User Library** dialog opens.
7. Specify a name for the new library and click **OK**. The **User Libraries** page opens.
8. Click **Add JARs**.
9. Browse to and select one or more Java libraries that are available to your local system.

   Press the `Shift` or `Ctrl` keys to select multiple libraries.
10. Click **Open** to add the JAR files to your new user library. The **User Libraries** page opens.
11. Click **OK**. The **Add Library** page opens.
12. Click **Finish**. These settings are migrated to the classpaths of both your Silk Performer Workbench project and your Silk Performer-enabled Eclipse project.

Adding a Silk Performer Test Class to an Eclipse Project

You can generate multiple Java classes for the testing of Silk Performer-enabled Eclipse projects.

Note: Silk Performer can handle user-defined JAR libraries. It can only launch classes that are within the root package of Java projects within Silk Performer. Only external libraries such as JARs can be used.

1. In the Eclipse SDK, choose **File** > **New** > **Other** . The **New** dialog box opens.
2. Choose **Silk Performer** > **Silk Performer Test Class** from the menu tree and then click **Next**. The **New Silk Performer Test Class** dialog box opens.
3. Edit the **Project name** and **Test class** values as required.
4. Click **[...]**. The **Silk Performer Enabled Projects** dialog box opens.
5. Select the Silk Performer-enabled Eclipse project to which you want to append this new test class template and then click **OK**.
6. Click **Finish**. The test class template is appended to your Eclipse project.

Try Script Runs

Executing Try Script Runs for Test Classes

1. In Eclipse's Package Explorer menu tree, select the test class for which you want to run a Try Script.
2. Select **Silk Performer 20.0** > **Try Script** . Eclipse automatically switches to the Debug perspective and initiates a Try Script run. The bottom pane of the Debug page includes Silk Performer views that monitor the status of the test.
3. If your test class generates errors or leads to a failed test, edit your test class in the Eclipse Editor.
4. If your Try Script run is successful, you can view the results of the test in the Silk Performer log files and result files, including TrueLog files.

Defining a Launch Configuration for Silk Performer Try Scripts

1. In Eclipse main toolbar, select **Run** > **Debug Configurations** . The **Debug Configurations** dialog opens.
2. Select the Silk Performer Try Script node in the menu tree.
3. Click the **New** button on the toolbar. The **Create, manage, and run configurations** page opens and the currently active Silk Performer project and its first test class are pre-selected.
4. Click **[...]** next to the **Project** box. The Silk Performer **Enabled Projects** dialog opens.
5. Select the Silk Performer-enabled project that you want to associate with this launch configuration.
6. Click **OK**.
7. Select the **Test class** that you want to launch with this launch configuration.
8. (optional) Edit the parameters of any pre-defined project attributes in the **Project Attributes** section of the page.
9. Click **Apply** to apply your changes.
10. Click **Debug** to run a Try Script run based on your launch configuration settings.

Running Baseline Tests

Once you are satisfied with the Try Script results generated by your TestClass, it is time to pass your test project from Eclipse to Silk Performer and run a baseline test.

1. In Eclipse, select **Silk Performer 20.0** > **Open in Silk Performer** . Silk Performer workbench launches with your Eclipse project loaded in the **Project** menu tree. All TestClasses are appended to the project under the Scripts node.
2. Define and run a baseline test.

Results

Following a successful test or Try Script run, you can choose to view a Virtual User Report file, a Virtual User Log file, a Virtual User Output file, a Virtual User Error File, or a TrueLog.

Viewing Virtual User Reports

In Eclipse, select **Silk Performer 20.0** > **Virtual User Report File** .

The report is then displayed in Eclipse's Silk Performer **Report File Viewer** tab. This view displays virtual-user status messages that are passed from the latest Try Script test.

Viewing Virtual User Log Files

In Eclipse, select **Silk Performer 20.0** > **Virtual User Log File** .

The log file of the most recent test run is displayed.

Viewing Virtual User Output Files

In Eclipse, select **Silk Performer 20.0** > **Virtual User Output File** .

The output file of the `TestClass` code for the most recent test run is displayed.

Viewing Virtual User Error Files

In Eclipse, select **Silk Performer 20.0** > **Virtual User Error File** .

The error file of the most recent test run is displayed.

Viewing TrueLogs

In Eclipse, select **Silk Performer 20.0** > **Show TrueLog** . The TrueLog file of the most recent Try Script run is then displayed in TrueLog Explorer.

Silk Performer Java Explorer

Silk Performer Java Explorer, which was developed using Java, and the Silk Performer Java Framework allow you to test Web Services, Enterprise JavaBeans (EJB), RMI objects, and other GUI-less Java objects. Java Explorer allows you to define and execute complete test scenarios with different test cases without requiring manual programming because tasks are completed visually by way of mouse-based operations. Test scripts are visual and easy to understand, even for individuals who are unfamiliar with Java programming languages.

Test scenarios created with Java Explorer can be exported to Silk Performer for immediate reuse in concurrency and load testing.

Note: Java Explorer is compatible only with JDK versions 1.2 and higher (v1.4 or higher is recommended).

For information about Java, visit the following Web sites:

- *http://java.sun.com*
- *http://www.javaworld.com*
- *http://www-106.ibm.com/developerworks/java/*

Starting Silk Performer Java Explorer

Perform one of the following steps to launch Java Explorer:

- Click **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Development Tools** > **Silk Performer Java Explorer** .
- Click **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Silk Performer Workbench** and create a new project with the `Java/Java Explorer` or `Web Services/Java Explorer` application type.

OnWeb Mainframe-to-Web Testing Support

Silk Performer supports the testing of Micro Focus OnWeb Web-based applications that interact with mainframe systems. OnWeb Web-based applications can easily be tested using standard Silk Performer Web-application recording, scripting, and playback features. When creating a script to test an OnWeb application, choose `Web business transaction (HTML/HTTP)` as the application type to be recorded. Beyond that, the testing of Micro Focus OnWeb Web-based applications is identical to that of standard Web applications. Silk Performer automatically handles any dynamic values that need to be parameterized. Additionally, using .NET Explorer and Java Explorer, OnWeb Web services can be tested in the same way that standard SOAP Web services are tested.

About Micro Focus OnWeb

Micro Focus OnWeb offers a comprehensive development and deployment platform that transforms your host applications into Web applications. OnWeb aggregates and composes data from multiple screens across multiple host systems and presents them as Web pages.

OnWeb® and its wide array of application connectors can help you bring together both host systems and enterprise applications to create standards-based Web-enabled composite applications, leveraging your existing IT investment and helping you move towards a service-oriented architecture (SOA).

Oracle Forms / Oracle Applications Support

This section explains how to use Silk Performer to run load tests against Oracle Forms and Oracle Applications. It briefly outlines possible server configurations and discusses how they can be tested. It also provides guidelines for customizing scripts, including verification functions.

Basic Concepts

This section explains Silk Performer's support for testing Oracle Forms and Oracle Applications, including a technology overview, project setup, and configuration issues.

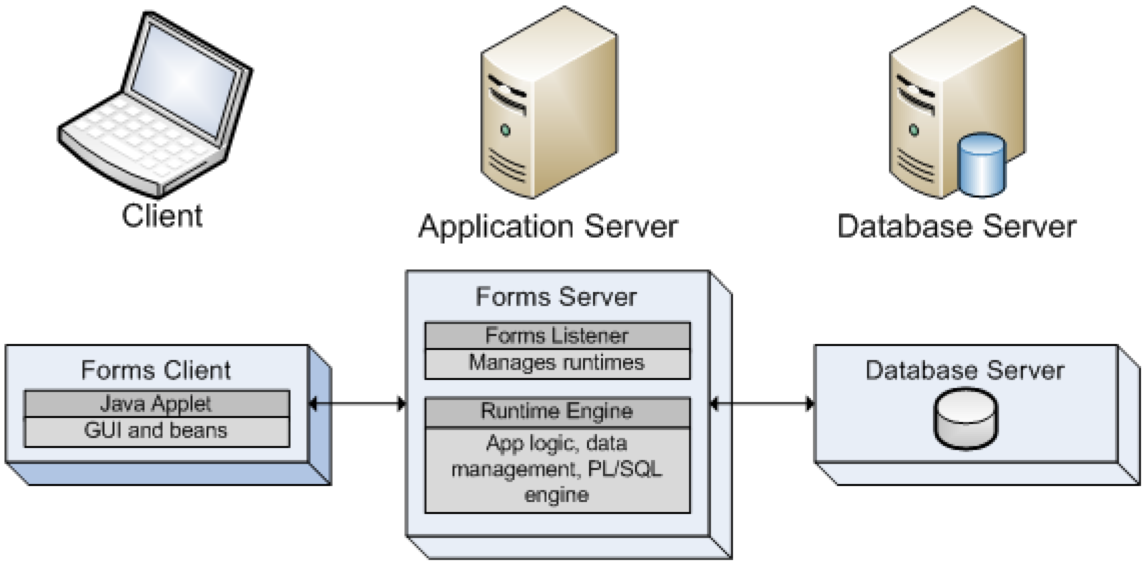
The following versions are supported:

- Oracle Forms 6i, 9i, 10g, 11g, 12c
- Oracle EBS 11i, 12.x
- Oracle Fusion 10g, 11g, 12c

Architecture

Oracle Web Forms allows you to run your existing Oracle Forms applications in a browser. In previous versions, the user interface and complete client/business logic where executed on the client machine in the Oracle Forms runtime. The move to Web technology split these components: There is still an Oracle Forms runtime that performs the client/business logic on an application server. However, the runtime has been separated from the user interface by sending/ receiving user interface actions to an applet that runs in a browser.

The following illustration offers a basic overview of the architecture:



When a client connects to an Oracle Forms server a HTTP-listener process returns the initial HTML page that contains information regarding the hosting of the applet. If a Java Virtual Machine (JVM) has not yet been installed, this page redirects your browser to download a JVM.

The applet connects to a listener process during initialization. This listener spawns a new process that hosts the runtime engine for the client. The client applet and the runtime engine then have a direct connection, via HTTP, HTTPS, or Socket mode.

Note: Socket mode is not available for Oracle Forms 6i patch 4, 10g, 11g, and 12c.

Oracle Forms 9i has introduced a test interface that can be used to retrieve all messages during recording and send those messages to the server during replay.

Oracle Forms Client Types

The Oracle Forms Web client is a Java applet hosted within a browser. Oracle Forms 11g or later uses the standard Java Virtual Machine (JVM), while for version 10g or earlier, Oracle provides its own JVM, which is called JInitiator.

A JVM or JInitiator is installed automatically when you access an Oracle Forms Web application for the first time. There are different versions of JInitiator, where the version numbers reflect the Java Runtime Environment version numbers that are used, for example JInitiator 1.1.8 uses Java Runtime Environment 1.1.8).

For detailed information on Oracle Forms and Oracle Applications, visit the *Oracle Technology Network*.

Oracle Forms/Oracle Applications Versions

There are different versions of Oracle Forms and, depending on the version you use, Silk Performer utilizes different approaches for recording. The following table illustrates the supported Oracle client versions and which Java version they use:

Oracle client version	Java version in Oracle client	Java version in Silk Performer profile settings
Oracle Forms 6	JInitiator 1.1.x	Java 1.4
Oracle Forms 9i	JInitiator 1.3.x	Java 1.4
Oracle Forms 10g	JInitiator 1.3.x	Java 1.4
Oracle Forms 11g	JVM 1.6	Java 1.6
Oracle Forms 12c	JVM 1.7	Java 1.7
Oracle Applications 11i (operates on Oracle Forms 6i)	JInitiator 1.1.x	Java 1.4
Oracle Applications 12i (operates on Oracle Forms 10g)	JVM 1.6	Java 1.6

Installation and Requirements

Oracle Forms applications require a Java Runtime Environment (JRE) on the client side. Silk Performer's record/replay mechanism also requires a JRE to test Oracle Forms applications.

Configuring Oracle Forms Client Software

Oracle Forms clients run in Web browsers as applets. By default, a Web browser will download and install a JRE when loading an Oracle Forms application for the first time. For Oracle Forms 10g or earlier, Oracle's JRE for Oracle Forms is named JInitiator. Using JInitiator is not mandatory. Once a JRE has been installed, Silk Performer is set up to record Oracle Forms applications.

Configuring Oracle Applications Server Software

Silk Performer Recorder modifies the HTML page hosting the Oracle Forms applet. It will set the parameter `record=names` so you will get readable names instead of IDs in your script. If this does not work automatically, proceed as follows:

1. On your Oracle Applications server, locate the file `formsweb.cfg` and open it with a text editor.
2. Append the text `record=names` to this file and save it.
3. Restart the Oracle Applications server.

Silk Performer is now able to record names instead of IDs.

Managing Oracle Forms/Oracle Applications Load Tests

This section explains the tasks that must be completed to prepare for, run, and analyze the results of an Oracle Forms/Oracle Applications load test.

Outlining Oracle Forms/Oracle Applications Projects

Before you begin testing an Oracle Forms application, ensure the following:

- The Oracle Forms client can be started on the recording machine.
- Know which version of Oracle Forms you are testing. Currently supported versions are Oracle Forms 6i, 9i, 10g, 11g, 12c and Oracle EBS 11i, 12.x.

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 the application type that matches your Oracle Forms/Oracle Applications version (**Application Server/Component Models** > **Oracle** > **Oracle Forms <version>**, or **ERP/CRM** > **Oracle** > **Oracle Applications <version>**).
5. Click **Next**.

   Note: If you need to add additional resources to the project, right-click the project icon in the **Project** menu tree view. It is particularly important that all the user data files (`.csv`), random data files (`.rnd`), and `.idl` files needed by Silk Performer are set up for your project.

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

Recording Oracle Forms/Oracle Applications Test Scripts

Record an Oracle Forms/Oracle Applications test script with Silk Performer Recorder before customizing it.

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. Select the default Oracle Forms recording profile. If you want to modify the settings of a custom recording profile, click **Settings**.
3. In the **URL** text box, enter the URL of your Oracle Forms application.
4. Click **Start recording** to begin recording the session.
5. To see a report of the actions that happen during recording, maximize the Recorder dialog by clicking the **Change GUI size** button. The maximized Recorder opens at the **Actions** page.
6. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.

7. Insert transactions and timers into the test script during the recording phase. You can create as many transactions and timers as you want. To insert a transaction, click the **New Transaction** button. A transaction represents a piece of work that can be assigned to a virtual user.
8. In the ensuing dialog, enter a name for the transaction and click **OK**. The new transaction appears in the **Actions** log.
9. To insert a timer, click the **New Timer Session** button. A timer is a user-defined measurement period in a test. You should create timers for each component of a transaction for which you want to analyze performance. In the ensuing dialog, enter a name for the timer and click **OK**.
10. To insert a timer, click the **New Timer Session** button. A timer is a user-defined measurement period in a test. You should create timers for each component of a transaction for which you want to analyze performance. In the ensuing dialog, enter a name for the timer and click **OK**.
11. To end recording, click the **Stop Recording** button.
12. Enter a name for the .bdf file and save it. The **Capture File** page displays. Click **Generate Script** to generate a script out of the capture file.

Recording Concepts

Oracle Forms applications are recorded by recording your browser and using a special Java Recording API that captures the messages that are sent between the client and server.

Recording Profile Settings

When choosing one of the Oracle Forms or Oracle Applications project types in your recording profile settings, a new recording profile is created for your project based on the active browser. You cannot modify the settings for this temporary recording profile. If you want to record using different settings, click **Copy** to create a custom recording profile based on the selected profile.

Oracle Forms 6 can operate on two different communication channels: HTTP/ HTTPS or Socket mode. When you select Oracle Forms 6, both channels are covered. If you encounter any problems during recording, select the correct Java Recording API in your recording profile (you must first determine which communication channel you are using).

Oracle Applications 12i will detect the communication channel (HTTP/ HTTPS or Socket mode) automatically.

Setting the Oracle Forms Log Level

The log level defines which messages and TrueLogs are captured when recording an Oracle Forms application.

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. Select **Record** in the shortcut list and click the **Oracle Forms** icon.
4. Click the **Logging** tab.

   Here you can specify that additional properties have their values logged during recording.
5. From the **Log level** list box, select the log level for virtual user logging:

   - `None` - TrueLogs will be generated, though no Oracle Messages will be logged and no detailed information about controls in the log file will be written.
   - `Error` - In addition to the `None` log level, errors that occur during recording are logged.
   - `Normal` - In addition to the `Error` log level, Oracle messages are logged to the TrueLog and the log file.

- `Debug` - In addition to the `Normal` log level, detailed information about control messages is logged, for example in the **In Body** and **Out Body** tab in TrueLog Explorer. This information is helpful for customizing and debugging your script when comparing it with Try Script runs. This option should also be used when you encounter a problem during replay and you must send your log files to Micro Focus SupportLine for analysis.

   Additional properties to be logged in the TrueLog for each control in your application can also be defined. Such properties are Oracle Forms internal properties that must be defined using their internal names. In most cases you will not need to use this feature, as default properties (name, value) are logged for each control. The `OraForms.bdh` file contains a complete list of all internal properties, most of which are not used by controls and so will generally be ignored if you define them.

6. Click **OK** to save your settings.

Oracle Java Recording API

The Java Recording API is comprised of two elements:

- HTTP recording rules
- Instructions regarding which Java classes implement recording logic

HTTP recording rules are necessary to prevent the recording of certain HTTP/ HTTPS traffic when HTTP/ HTTPS is the transport protocol of the client/server communication channel. Not all HTTP/HTTPS traffic is ignored-only the traffic that sends/receives messages is ignored.

This means that if your Oracle Forms application is embedded in a normal HTML file with embedded objects or frames, the calls that request those pages will be scripted and therefore they will be requested during replay.

In terms of scripting the necessary `OraForms` API calls, all communication between the applet and the server is handled by the Java Recorder.

Java Environment Settings

As recording and replay of Oracle Forms applications is achieved through a Java implementation, it is necessary to configure your Java environment in Silk Performer, using the **Profile** settings dialog. You must define your Java home directory (**Settings** > **Active Profile** > **Java** > **General** > **Java home**).

Files Generated During Recording

When recording an application, a script, a text log and a TrueLog are generated.

Script

The script contains the `OraFormsInit` method call in the `init` transaction. `OraFormsInit` defines the server, port, servlet URL and version of your Oracle Forms application. Not all parameters will necessarily be scripted with values as required parameters vary based on version and connection type.

`OraFormsInit` initializes the internal Oracle Forms replay engine (the actual connect is achieved with `OraFormsConnect`) and therefore should be one of the first calls in the main transaction.

Wherever a new form window is activated the recorder scripts a `OraFormsSetWindow`, indicating that the new window is now the active window. `OraFormsSetWindow` has a special meaning in TrueLogs, which will be discussed later in this chapter.

All of the subsequent method calls until the next `OraFormsSetWindow` are executed on controls of the current window.

`OraFormsDestroy` is scripted in the `end` transaction to shut down the replay engine.

Recording Text Log

The recording text log contains information about the HTTP traffic.

Most of the HTTP information that would normally be logged for a Web application is disabled. If you are interested in detailed HTTP logs, enable this setting in your profile.

The relevant elements of the log are the Oracle Forms messages that are logged when you have a log level of `Normal` or `Debug`. Messages from and to the server are logged in the following format:

Sample Message	Description
===>	Direction: ===> (to server), <== (from server)
Block# 1 --------------------	Block #: A block contains a message sent to the server and the server's response to the message. Both parts are terminated by a terminal message.
MSGTYPE: UPDATE	The type of message. Options include: -CREATE, UPDATE, GET, DESTROY, TERMINAL
CLASS: 1/1	The class of control that handles the message
ID: 1	The unique control identifier that handles the message
RESPONSE: 0	Status message
PROPERTIES: TYPE:PROP_TYPE_INTEGER Name:INITIAL_VERSION Value:90290 TYPE:PROP_TYPE_POINT Name:INITIAL_RESOLUTION Value:java.awt.Point[x=96,y=96] TYPE:PROP_TYPE_POINT Name:INITIAL_DISP_SIZE Value: java.awt.Point[x=1280,y=1024] ...	A list of properties. Each property has a type, name, and value.

When the log level is set to `Debug`, log information required for troubleshooting is logged to the log file. If you encounter any problems and need to contact Micro Focus SupportLine, please provide a log file utilizing the log level `Debug`.

Record TrueLogs

Record TrueLogs contain the same information as replay TrueLogs. This allows you to compare record and replay differences after you have executed your first TryScript.

Oracle Forms HTTPS Support

Silk Performer enables you to record Oracle Forms applications that use secure connections via HTTPS. To record an Oracle Forms application over HTTPS you must add the Silk Performer certificate to your JVM’s truststore database:

Enabling Oracle Forms HTTPS Support Using a JVM

Silk Performer enables you to record Oracle Forms applications that use secure connections via HTTPS. To record an Oracle Forms application over HTTPS you must add the Silk Performer certificate to your JVM’s truststore database:

1. Open the **Java Control Panel** on your system.
2. Select the **Security** tab.
3. Click **Import** to locate and import the Silk Performer certificate `IRCAcert.pem`. This file is located in your Silk Performer installation directory.

4. Close the **Java Control Panel**.

   Note: Remember to start a new browser session before recording your Oracle Forms application.

You can now record your application.

Enabling Oracle Forms HTTPS Support Using JInitiator

Silk Performer enables you to record Oracle Forms applications that use secure connections via HTTPS. To record an Oracle Forms application over HTTPS you must add the Silk Performer certificate to your JVM's truststore database:

1. Open the file `certdb.txt` with a text editor. This file is located in the directory `\Lib\security\` of your JInitiator's home directory.
2. Open the Silk Performer certificate file `IRCAcert.pem` with a text editor. This file is located in your Silk Performer installation directory.
3. Copy the content of the Silk Performer certificate file and append it to the content of the `certdb.txt` file.
4. Save and close the `certdb.txt` file.

   Note: Remember to start a new browser session before recording your Oracle Forms application.

You can now record your application.

Oracle Forms / Oracle Applications TrueLogs

TrueLog Explorer is a powerful tool that not only offers a convenient means of exploring Oracle Forms applications. It also enables you to add verifications for control values and customize input values for controls that are modified during recording.

Working With Oracle Forms Applications - Overview

Oracle Forms, previously called "SQL*Forms", is part of Oracle's Internet Developer Suite (iDS). It is a 4GL Rapid Application Development (RAD) environment that allows forms to be deployed across the Web via Oracle's Internet Application Server (iAS) Forms Services.

Because Oracle Forms is based on Java technology, before you can record and replay Oracle Forms transactions, you must configure Java Virtual Machine using Silk Performer profile settings. The Java Just-In-Time Compiler must also be disabled while recording Oracle Forms 6i or higher.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

Oracle Forms TrueLog Structure

This section explains the Oracle Forms TrueLog structure and the **Form Controls** window.

Working With Web Calls

In addition to Oracle Forms API calls, Web calls that download HTTP content are also included in TrueLogs. The first call in each main transaction is the Web call that downloads the initial page of the Oracle Forms application under test.

The TrueLog Explorer feature set varies based on the protocol of the open TrueLog. Because Oracle Forms is a mixed protocol that often incorporates Web calls in scripts, TrueLog Explorer enables you to alternate between a tabular data-based representation view for binary Oracle Forms applets and a rendered HTML view for HTML pages. The Web-based protocol is significant because it enables you to customize session handling, insert verification functions, parse values out of HTML, and run searches on HTML included in Oracle Forms scripts.

1. Choose **Edit** > **TrueLog Type** .
2. Select **Web** (default).

Node Information

Each Oracle Forms node stores information about the current state of all controls of the active window. If you select a `OraFormsSetWindow` node that represents a window's first appearance, you will see all the controls and initial values as they existed when the form was initially created.

If you select a subnode, you will see the state and values of all controls as they existed after the action you selected was fulfilled. For example, if you select an `OraFormsEditSet` you will see the state of the controls after this action was fulfilled.

If a window is reactivated (meaning that an alternate window was activated in the meantime) and you select the `OraFormsSetWindow` of the reactivated window, you will see the status of the controls as they were after the reactivation (which in most cases is the same status they had when the window was deactivated).

Verification Checks with TrueLog Explorer

TrueLog Explorer enables you to insert content-verification functions into test scripts to verify the accuracy of content that is returned by application servers during testing.

TrueLog Explorer offers wizards that add verification functions to your scripts to verify control values at any time during an application's life cycle. To add a verification function, right-click in the **Value** column of a control that you wish to verify and select the relevant context menu.

When you identify the content that you want to verify, all required verification functions can be generated and automatically inserted into your test script. To identify content that is to be verified (within rendered HTML, HTML source code, SQL commands, Oracle Forms, or elsewhere), select and right-click it.

Verifications can be applied visually inTrueLog Explorer **Rendered** and **Source** views using any of the following methods:

- **Script** menu
- **Add Verifications** dialog box
- Context menus within **Rendered**, **Source**, and **Form Controls** views
- workflow bar

Parsing Functions Overview

Parsing functions are typically used for the following tasks:

- Replacing static session IDs in scripts with dynamic session IDs that maintain state information.
- Building enhanced content verifications into scripts that can not be achieved with verification functions alone. For example, a parsing function might verify that a value in column 2 of row 3 of a database table is equal to the sum of the values in column 2 of row 1 and column 2 of row 2. This can be achieved by generating parsing functions that parse out the three values and compare them in a script.
- Conditionally executing part of a testing script that is dependent on data returned from a server. For example, an HTTP request returns an HTML page that includes the following results: <nnn> `items found`. Different actions need to be executed against the value <nnn>. Say the transaction is designed to:
    - Exit if <nnn> `= 0`.
    - Link to a details page if <nnn> `= 1`.
    - Link to the next page if <nnn> `is greater than 1`.

To accomplish this, the value <nnn> must be parsed from the HTML page, and scripted actions must be run based on the parsed values.

TrueLog Explorer allows you to insert parsing functions visually in **Rendered** and **Source** views. TrueLog Explorer automatically generates parsing functions in scripts, so no manual writing of code is required.

TrueLog Explorer offers wizards that add parsing functions to your scripts to parse control values at any time during an application's life cycle. To add a parsing function, right-click in the **Value** column of a control that you wish to parse and select the relevant context menu.

Input Data Customizable Functions

TrueLog Explorer offers a wizard that enables you to customize the input values of controls that have been changed by the user during recording. Input values can be customized at the same position where input originally occurred during recording. Here is a list of the functions that can be customized:

Function	Description
`OraFormsEditSet`	Customizes the value of a text control.
`OraFormsRadioSet`	Specifies whether a radio button is to be selected.
`OraFormsCheckboxSet`	Specifies whether a check box is to be selected.
`OraFormsListSelect`	Specifies which element of a list box is to be selected.
`OraFormsPopListSelect`	Specifies which element of a pop-up list box is to be selected.
`OraFormsLogon`	Customizes logon credentials on the Logon dialog.
`OraFormsLovFind`	Customizes the search pattern on a List Of Values dialog.
`OraFormsLovSelect`	Customizes the selection on a List Of Values dialog.
`OraFormsEditorDialogOK`	Customizes a text control value in an editor dialog.

In Body / Out Body Pages

The data that is communicated between Silk Performer and an Oracle Forms server can be tracked on the In Body and Out Body pages. During Oracle Forms replay, the In Body page displays the data that is received by Silk Performer from the Oracle Forms server. The Out Body page displays the data that is sent by Silk Performer to the Oracle Forms server.

Note: These messages are only available when you set the logging options in the **Profile Settings** dialog to **Debug**.

Messages can be ignored during data customization (except when customizing GET messages). However if a problem arises and Customer Care is contacted, this information will be required for analysis. You may notice major differences between record and replay TrueLogs. All messages are likely to be included, but they may not be logged at the corresponding nodes; they may be logged one node earlier or later.

Each message block (`OraForms` call) that is communicated between Silk Performer and the Oracle Forms server is comprised of one or more of the following messages types:

Item	Description
Create	Indicates that a new UI element must be created. Each UI object to be created is given a class, properties, and an ID. Example: The server tells the client to create a text box of a given type, with a given name, and a given ID, at a given location.
Destroy	Indicates that a UI element must be destroyed. Example: When a window is closed, all of the controls on the window must be destroyed.
Update	Indicates what an element within the applet should look like (for example, position, focus, and selection status).

Item	Description
	Example: The client tells the server where the mouse cursor should be positioned. The server then responds by changing the mouse cursor position within the applet's interface.
Get	Indicates internal communication between the client and server. Get messages are not visible in the UI. Example: The server asks the client what the ID of a certain control is. Get messages are typically terminated with Terminal 3 messages.
Terminal	Indicates the end of a communication round trip. Each message block is terminated with a terminal message. Three terminal types are available: • Terminal 1 • Terminal 2 • Terminal 3

The most common message block type involves the client sending an update message to the server and ending the communication with a Terminal 1 message. The server then typically responds with a create or destroy message that is terminated by a terminal 1 message.

Example In Body/Out Body message block:

With a mouse click function, the client sends an update message to the server with some properties. One of the properties is the mouse cursor location. Another property is that the mouse should be in the pressed-down state. The round trip of the communication is then closed with a Terminal message. The server then responds, indicating that the cursor has been repositioned and set to the pressed-down state.

```
MSGTYPE:       UPDATE
CLASS:         0/0
ID:            2824
TITLE:         N/A
RESPONSE:      0
PROPERTIES:
TYPE:  PROP_TYPE_INTEGER
Name:  MENU_MENUUPDATE/367
Value: 2855

MSGTYPE:       DESTROY
CLASS:         0/0
ID:            2855
TITLE:         N/A
RESPONSE:      0
PROPERTIES:

MSGTYPE:       GET
CLASS:         0/0
ID:            1644
TITLE:         N/A
RESPONSE:      0
PROPERTIES:
TYPE:  PROP_TYPE_VOID
Name:  VALUE/131
Value: null

MSGTYPE:       TERMINAL
CLASS:         0/0
ID:            0
```

```
TITLE:          N/A
RESPONSE:       3
PROPERTIES:
```

Replay Concepts

Silk Performer emulates multiple Oracle Forms clients distributed across multiple Silk Performer agents. Silk Performer's Oracle Forms clients are driven by a Java Runtime that is hosted by Silk Performer during replay.

Setting Oracle Forms Options

Oracle Forms-specific settings allow you to modify how Silk Performer interacts with Oracle Forms clients during replay.

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list, click the **Oracle Forms** icon.
4. On the **General** page, select a **Connection mode** setting that reflects which type of connection the Oracle Forms server accepts.

   Oracle Forms clients can communicate with servers via `Sockets`, `HTTP`, or `HTTPS`.

   Note: This setting can be overridden by the `OraFormsSetConnectMode(connectMode)` BDL function call.

   - Click the **Socket mode** option button when the Oracle Forms server is set up for socket mode.
   - Click the **HTTP mode** option button for Oracle Forms servers that are configured for HTTP connections.
   - Click the **HTTPS mode** option button for Oracle Forms servers that are configured for secure HTTP connections.
5. In the **Connection timeout** text box, specify a timeout period (in seconds) to specify how long emulated clients should attempt to establish communication with the server before they report an exception.
6. To configure additional runtime settings, you can optionally have the virtual user send a heartbeat message to the server at a specified interval. This is useful when the Oracle Forms client is not communicating with the server for a long period of time, for example during a long think time period. Check the **Enable heartbeat with frequency of** check box and enter an interval (in seconds) in the **sec** field.
7. Check the **Automatically wait for application timers** check box to direct the replay engine to wait for application timers to expire after each function call.
8. The **Application timer and window timeout** text box specifies the maximum wait time for expiration of application timers and appearance of windows. Enter a timeout setting (in seconds).

   The replay engine waits for timers to expire after each function call. The function `OraFormsWaitForTimer` can be used to make an application wait for timers to expire. Maximum wait time is specified by this setting. The function `OraFormsWaitForWindow` may be used to have an application wait for windows to appear. This setting specifies the maximum wait time for window appearance.
9. Oracle Forms servers can allow certain client releases. Use the Oracle Forms setting to specify the Oracle Forms client version.

   Oracle Forms clients send the required version when connecting to the Oracle Forms server. Create profiles and change this setting when testing applications deployed on different server versions.

Note: This setting can be overridden by the BDL function call `OraFormsSetInt("INITAL_VERSION", theVersion)`.

10. Click the **Logging** tab.

    Here you can specify that additional properties have their values logged during replay.

11. From the **Log level** list box, select the log level for virtual user logging:

    - `None` - TrueLogs will be generated, though no Oracle Messages will be logged and no detailed information about controls in the log file will be written.
    - `Error` - In addition to the `None` log level, errors that occur during replay are logged.
    - `Normal` - In addition to the `Error` log level, Oracle messages are logged to the TrueLog and the log file.
    - `Debug` - In addition to the `Normal` log level, detailed information about control messages is logged, for example in the **In Body** and **Out Body** tab in TrueLog Explorer. This information is helpful for customizing and debugging your script when comparing it with Try Script runs. This option should also be used when you encounter a problem during replay and you must send your log files to Micro Focus SupportLine for analysis.

    Additional properties to be logged in the TrueLog for each control in your application can also be defined. Such properties are Oracle Forms internal properties that must be defined using their internal names. In most cases you will not need to use this feature, as default properties (name, value) are logged for each control. The `OraForms.bdh` file contains a complete list of all internal properties, most of which are not used by controls and so will generally be ignored if you define them.

12. In the **Additional properties** text box, you can specify other properties for which virtual users are to log values.

    - Click **Add** to open the **Additional Properties** dialog box and add a property that is to have its value logged during replay.
    - Click **Edit** to open the **Additional Properties** dialog box and edit the selected property.
    - Click **Remove** to remove a selected property. Click **Yes** on the deletion confirmation dialog. No further values will be logged for this property.

13. Click the **Measuring** tab.

    Here you can specify a custom measurement level. By default, all available performance metrics are collected during test runs.

14. Check the **Enable all timers and counters for all controls** check box to enable all timers and counters for all actions on controls during replay.

    Alternatively, uncheck the **Enable all timers and counters for all controls** check box and select a specific control from the **Control** list box and select the specific timer/counter types that you want to have applied to that control:

    - `Enable timers` - Measure how long it takes to complete an action on the selected control.
    - `Count round trips` - Count the round trips for each action on the selected control.
    - `Count bytes` - Count the bytes sent and received for each action on the selected control.
    - `Count messages` - Count the messages sent and received for each action on the selected control.

    Note: Alternatively, you can click the **Apply to All Controls** button to apply your timer/counter settings to all controls.

15. Click **OK** to save your settings.

Oracle Forms Client Sessions

When executing a recorded Oracle Forms script, a client session begins with the `OraFormsConnect` function call and ends with `OraFormsDisconnect` or `OraFormsDestroy`.

Example

```
transaction TInit
begin
```

```
  WebSetBrowser(WEB_BROWSER_MSIE8);
  WebModifyHttpHeader("Accept-Language", "en,de-at;q=0.5");
  OraFormsInit("http://lnz-vmoraf11:8888/forms/lservlet", " lnz-
vmoraf11", 9000,
  ORA_FORMS_11G);
end TInit;

transaction TShutdown
begin
  OraFormsDestroy();
end TShutdown;

transaction TMain
var
begin
  OraFormsSetString("DEFAULT_LOCAL_TZ", "Europe/Berlin");
  OraFormsConnect(
    "server module=healthyliving.fmx userid=hl/hl@ORCL_ SERVER
sso_userid="
    "output_dir=C:\\orant\\forms11\\demos\\temp");

  // New window activated: MAIN
  ThinkTime(3.7);
  OraFormsSetWindow("MAIN");
  OraFormsEditSet("EDIT_CONTROL_0", "USER",
ORA_EDIT_SEND_DIRTY); // Supplier Name
  OraFormsDisconnect();
end TMain;
```

The above session begins with the `OraFormsConnect` call and ends with the end of the transaction (`OraFormsDisconnect`). Therefore each iteration of the transaction can be considered an Oracle Forms client session.

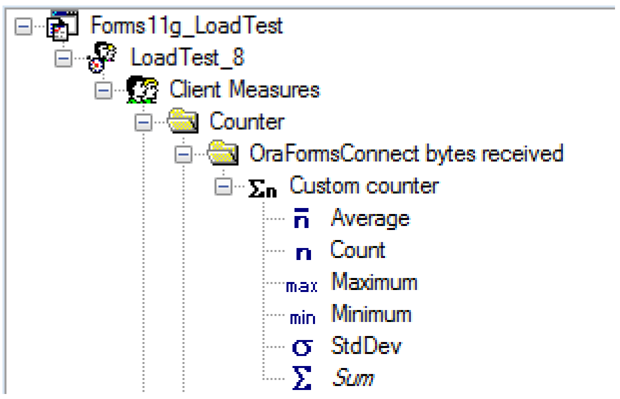
Results

Once a Silk Performer test is complete, Performance Explorer is used to analyze how the Oracle Forms application performed during the test run. Silk Performer automatically collects a configurable number of timers and counters for each function call, depending on which performance metrics you specified to be collected during test runs.

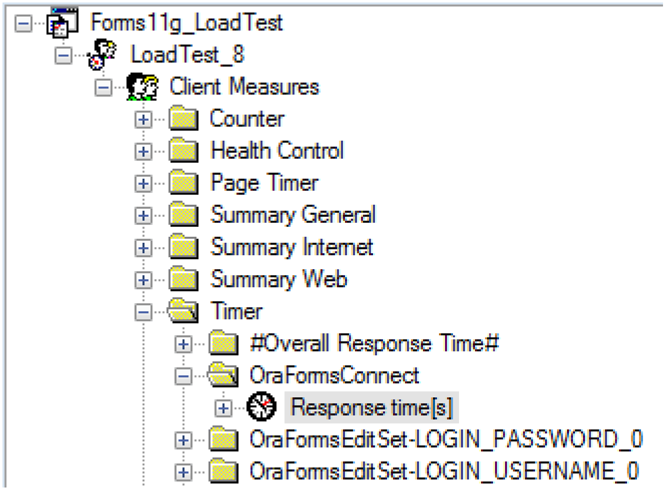
By default, Silk Performer collects the following counters for each function call:

- bytes sent
- bytes received
- messages received
- messages sent
- roundtrips

These counters are found in Performance Explorer as follows:



In addition to counters, Silk Performer offers timers, which measure how long it takes for a function call to complete its tasks. These timers are found in Performance Explorer as follows:



Graphs can be assembled by dragging counters and timers into graphs.

Connection Handling

Oracle Forms connections are established with `OraFormsConnect` and are closed with `OraFormsDisconnect`. Recorded scripts include `OraFormsConnect` at the beginning of the `TMain` transaction and `OraFormsDisconnect` as the last Oracle Forms call in `TMain`. `OraFormsDisconnect` must be called to ensure that the server connection is closed, otherwise the server runtime process that handles the client will continue to run.

`OraForms.bdh` implements event handlers that call `OraFormsDisconnect` when server errors cause virtual user process exits.

In scenarios in which you have a login procedure and multiple main transactions that perform actions that you want to test, you must ensure that you have established connections at the beginning of transactions. As connections may close due to severe errors, subsequent transactions must re-establish connections.

The suggested means of handling this issue is to create a BDL function that contains the script code that establishes the connection to the Oracle Forms application. The function can use the `OraFormsIsConnected` method to query if a connection is already established. This function should be called at the beginning of each Oracle Forms transaction that requires a connection to the application.

Example

After recording, your script should resemble the following:

```
var
  gsSSessionID : string;

dclrand

dcltrans
  transaction TInit
  begin
    OraFormsInit("http://lnz-vmoraf11:8888/forms/lservlet",
"lnz-vmoraf11", 9000, ORA_FORMS_11G);
    //WebSetUserBehavior(WEB_USERBEHAVIOR_FIRST_TIME);
    //WebSetDocumentCache(true, WEB_CACHE_CHECK_SESSION);
  end TInit;
  transaction TShutdown
  begin
    OraFormsDestroy();
  end TShutdown;

  transaction TMain
  begin
    WebSetBrowser(WEB_BROWSER_MSIE8);
    WebModifyHttpHeader("Accept-Language", "de-at");
    WebPageUrl("http://lnz-vmoraf11:8888/forms/frmservlet",
"iOrganizer", FORMS_FRMSERVLET001);

    WebSetBrowser(WEB_BROWSER_CUSTOM);
    WebSetUserAgent("Java1.6.0.21");
    WebSetHttpVersion("HTTP/1.1");
    WebModifyHttpHeader("Accept-Language", NULL, WEB_MODIFY_
OPT_Remove);
    WebUrl("http://lnz-vmoraf11:8888/forms/java/f90all_
jinit.jar", 29.61);
    WebUrl("http://lnz-vmoraf11:8888/forms/jars/
iorganizer.jar", 0.42);
    WebUrl("http://lnz-vmoraf11:8888/forms/iorg_images/
iorganizer.gif", 0.45);
    WebUrl("http://lnz-vmoraf11:8888/forms/images/blue.gif",
0.43);
    WebUrl("http://lnz-vmoraf11:8888/forms/java/oracle/forms/
registry/Registry.dat", 0.08);
    WebUrl("http://lnz-vmoraf11:8888/forms/iorg_registry/
iorg_registry.dat", 6.58);

    // Connect - with connection properties
    OraFormsSetInt("INITIAL_VERSION", 1111003);
    OraFormsSetPoint("INITIAL_RESOLUTION", 96, 96);
    OraFormsSetPoint("INITIAL_DISP_SIZE", 1280, 1024);
    OraFormsSetInt("INITIAL_COLOR_DEPTH", 256);
    OraFormsSetString("FONT_NAME", "Dialog");
    OraFormsSetInt("FONT_SIZE", 900);
    OraFormsSetByte("FONT_STYLE", 0);
    OraFormsSetByte("FONT_WEIGHT", 0);
    OraFormsSetPoint("INITIAL_SCALE_INFO", 8, 20);
    OraFormsSetBoolean("WINSYS_REQUIREDVA_LIST", false);
    OraFormsSetString("DEFAULT_LOCAL_TZ", "Europe/Berlin");
    OraFormsConnect("server escapeParams=true
module=iorganizer.fmx userid=  debug=no host= port= usesdi=yes
record=names",
      "http://lnz-vmoraf11:8888/forms/lservlet?ifcfs=/forms/
frmservlet?config=iorg&ifsessid=formsapp.4&acceptLanguage=de-
at",
      "http://lnz-vmoraf11:8888/forms/frmservlet?config=iorg");
    OraFormsGetSessionId(gsSSessionID);
    WebCookieSet(gsSSessionID, "http://lnz-vmoraf11:8888/forms/
java/");

    // ---
    // New window activated: 10
    OraFormsSetWindow("10");
    ThinkTime(6.6);

    …
end TMain
```

After customization, your script should appear as follows:

```
var
  gsSSessionID : string;

dclfunc
  function DoConnect
  begin
    WebSetBrowser(WEB_BROWSER_MSIE8);
    WebModifyHttpHeader("Accept-Language", "de-at");
    WebPageUrl("http://lnz-vmoraf11:8888/forms/frmservlet",
"iOrganizer", FORMS_FRMSERVLET001);

    WebSetBrowser(WEB_BROWSER_CUSTOM);
    WebSetUserAgent("Java1.6.0.21");
    WebSetHttpVersion("HTTP/1.1");
    WebModifyHttpHeader("Accept-Language", NULL, WEB_MODIFY_
OPT_Remove);
    WebUrl("http://lnz-vmoraf11:8888/forms/java/f90all_
jinit.jar", 29.61);
    WebUrl("http://lnz-vmoraf11:8888/forms/jars/
iorganizer.jar", 0.42);
    WebUrl("http://lnz-vmoraf11:8888/forms/iorg_images/
iorganizer.gif", 0.45);
    WebUrl("http://lnz-vmoraf11:8888/forms/images/blue.gif",
0.43);
    WebUrl("http://lnz-vmoraf11:8888/forms/java/oracle/forms/
registry/Registry.dat", 0.08);
    WebUrl("http://lnz-vmoraf11:8888/forms/iorg_registry/
iorg_registry.dat", 6.58);

    // Connect - with connection properties
    OraFormsSetInt("INITIAL_VERSION", 1111003);
    OraFormsSetPoint("INITIAL_RESOLUTION", 96, 96);
    OraFormsSetPoint("INITIAL_DISP_SIZE", 1280, 1024);
    OraFormsSetInt("INITIAL_COLOR_DEPTH", 256);
    OraFormsSetString("FONT_NAME", "Dialog");
    OraFormsSetInt("FONT_SIZE", 900);
    OraFormsSetByte("FONT_STYLE", 0);
    OraFormsSetByte("FONT_WEIGHT", 0);
    OraFormsSetPoint("INITIAL_SCALE_INFO", 8, 20);
    OraFormsSetBoolean("WINSYS_REQUIREDVA_LIST", false);
    OraFormsSetString("DEFAULT_LOCAL_TZ", "Europe/Berlin");
    OraFormsConnect("server escapeParams=true
module=iorganizer.fmx userid=  debug=no host= port= usesdi=yes
record=names",
      "http://lnz-vmoraf11:8888/forms/lservlet?ifcfs=/forms/
frmservlet?config=iorg&ifsessid=formsapp.4&acceptLanguage=de-
at",
      "http://lnz-vmoraf11:8888/forms/frmservlet?config=iorg");
    OraFormsGetSessionId(gsSSessionID);
    WebCookieSet(gsSSessionID, "http://lnz-vmoraf11:8888/forms/
java/");

  end DoConnect;

dcltrans
  transaction TInit
  begin
    OraFormsInit("http://lnz-vmoraf11:8888/forms/lservlet",
"lnz-vmoraf11", 9000, ORA_FORMS_11G);
    //WebSetUserBehavior(WEB_USERBEHAVIOR_FIRST_TIME);
    //WebSetDocumentCache(true, WEB_CACHE_CHECK_SESSION);
  end TInit;
```

```
transaction TShutdown
begin
  OraFormsDestroy();
end TShutdown;

transaction TMain
begin
  DoConnect();

  // ---
  // New window activated: 10
  OraFormsSetWindow("10");
  ThinkTime(6.6);
  …
end Tmain

transaction TsecondMain
begin
  DoConnect();

  // ---
  // New window activated: 10
  OraFormsSetWindow("10");
  ThinkTime(6.6);
  …
end TsecondMain;
```

With this approach, if you run a load test with a user that executes multiple main transactions you will not experience problems with subsequent transactions if an active transaction fails and disconnects from the server.

Oracle Forms Performance Monitoring

Use Performance Explorer to retrieve performance metrics from Oracle Forms Dynamic Monitoring System (DMS). Performance Explorer can monitor the DMS performance Web page that is provided through the Oracle Forms HTTP server when the DMS module is installed.

For full details on Oracle Forms performance monitoring, please see Performance Explorer Help.

Root Cause Analysis

Errors that happen during a Try Script, baseline or load test-run can easily be analyzed with the log files that are written during the run - especially with the TrueLog.

TrueLog Analysis

The most common error that occurs is: `OraForms: 5 - Handler not found`.

This error is thrown if the control (handler) that is referenced as the first parameter of a failing `OraForms` API call cannot be found, for example a button that should be pressed is not on the form.

By having a look at the preceding calls - prior to the one that throws the error - it is often easy to see why the control is not there. Oracle Forms applications often send a `WINSYS_BEEP` message if an error occurs, for example when entering invalid data. Some applications also update the status bar message. Moving through the preceding calls and looking at the **Info** tab will show you if there was a `WINSYS_BEEP` or a status bar update.

Very often, the server updates the status bar message in case of an error. Check the preceding calls, prior to the one that throws the error, if there is any information about a changed status bar message. Whenever the status bar message changes, an informational log entry will be logged.

Log File Analysis

The log files (recording and replay) contain different information depending on the log level defined in the profile settings. Whenever you have a problem and need to contact technical support, switch the log level to `Debug` and try to record/replay the script again and send those files. The `Debug` log level contains special development information that allows Technical Support to analyze the problem quicker.

The log file contains the Oracle Forms messages that are sent between the server and the client. Additionally, in debug log-mode all client side events will be logged, for example value changed, mouse down, and more.

OraForms Errors

Below is a list of the `OraForms` errors that can be thrown by the `OraForms` API calls.

ID	Message	Description
1	Logon dialog not found	`OraFormsLogon` and `OraFormsLogonCancel` can throw this error if the **Logon** dialog is not the current active dialog. It is most likely that the `OraFormsConnect` failed because the logon dialog should be the first dialog after a successful connect.
2	Window not found	`OraForms-CloseWindow`, `-WindowMove`, `-WindowResize` and `-WindowUpdate` will throw this error if the window with the passed window name/ID has not yet been created. Similar to the `5 - handler not found` error message.
3	Unsupported property	Can be thrown by multiple `OraForms` calls when the calls are querying handlers for special properties that should be set but are not. This error is not likely to happen under normal circumstances.
4	Focus handler not found	`OraFormsSetFocus` throws this error if the handler with the passed handler name/id was not found. Similar to the `5 - handler not found` error message.
5	Handler not found	This method can be thrown by any `OraForms` API call that takes a `handler name/ID` as first parameter. The first thing that every API call verifies is if the handler with the passed name/ID has already been created. If not, the operation cannot be executed on that handler. There are multiple reasons why a handler is not present during replay although it was during recording. The most common case is that an error was raised on the server, for example because of invalid user input. This invalid input has caused the application to not display dialogs or other windows that would normally appear when entering valid data like it does during recording.
6	The button to be pressed does not exist	`OraFormsMessageBoxButton` throws this error if an invalid button number was passed to the method. Valid buttons are: `ORA_ ALERT_YES`, `ORA_ALERT_NO`, `ORA_ALERT_CANCEL`.
7	Menu item was not found	`OraFormsMenuItem` throws this error if the menu item passed by its full name was not found. The menu item is searched in the assigned menu of the passed window name/ID.
8	Tab page not found	`OraFormsTabSelect` throws this error if there is no tab page with the passed name or passed index.

ID	Message	Description
9	Unrecognized dialog button	`OraFormsCloseDialog` throws this error if an invalid button number was passed to the method. Valid buttons are: `ORA_DIALOG_OK` and `ORA_ DIALOG_CANCEL`.
10	The function cannot be applied for this handler	Several `OraForms` API calls throw this error if the method was called on a handler of a type that does not allow to execute that kind of function,for example `OraFormsTabSelect` on a non-tab control handler.
11	The LOV row does not exist	`OraFormsLovSelect` throws this error if there is no entry in the current list of values dialog that matches the passed name or passed row index.
12	The list item does not exist	`OraFormsListSelect` and `OraFormsListActivated` throw this error if the list handler referenced in the first parameter does not contain a list element with the passed value or index.
13	Handler is of invalid class	Several `OraForms` API calls throw this error if the method was called on a handler of a type that does not allow to execute that kind of function, for example `OraFormsTabSelect` on a non-tab control handler.
14	A value for this property is not set	All `OraFormsGetProp{Type}` methods throw this error if the requested property has not been set on the passed handler.
15	Array index is out of bound	`OraFormsGetPropStringArray` and `OraFormsGetPropByteArray` throw this error if the passed element index is out of the array boundaries of this property. Use `OraFormsGetPropStringArrayLen` and `OraFormsGetPropByteArrayLen` to get the number of elements in the array. The index is 0-based.
16	Verification failed	Every verification function throws this error if the defined verification fails.
17	Received version too old	If the server returns the error `VERSION_TOO_OLD` during the connect phase, this error is thrown. This means that the version number of the client is too old to work with the version on the server. Use a different version number for the client by customizing the `OraFormsSetInt` for the property `INITIAL_VERSION` or the version number in the profile settings.
18	Received version too new	If the server returns the error `VERSION_TOO_NEW` during the connect phase, this error is thrown. This means that the version number of the client is too new to work with the version on the server. Use a different version number for the client by customizing the `OraFormsSetInt` for the property `INITIAL_VERSION` or the version number in the profile settings.
19	Alert dialog	Whenever an alert dialog is created, this error is thrown. The default severity of this error is `INFORMATIONAL` as alert dialogs are often used as the normal way to display a message to a user.
20	Statusbar changed	Whenever the server updates the status bar text, this error is raised with the new text of the status bar. The default severity is `INFORMATIONAL` as a status bar update is normally a way to inform the user about server side actions, but it could

ID	Message	Description
		also be an error message. Therefore, when doing root cause analysis of errors check for status bar changes.
21	Runtime error, see log file for stack trace	If there are internal replay runtime errors, this error is logged with a detailed stack trace in the replay log file. In case of such an error, contact technical support and send all recorded and replayed files.
22	WINSYS_BEEP	This error is logged if the server sends a `WINSYS_BEEP` message. Oracle Forms applications send a `WINSYS_BEEP` to indicate that the last action caused some problems on the server, for example invalid user input. It is likely that a `WINSYS_BEEP` is followed by other errors like `Handler not found`. The reason for this is because a recorded session without errors could cause errors during replay if the script is customized with invalid input values, or if a script that should be customized is not customized, for example using different values each time a test is run.
23	`OraFormsSetConnectMode` overrides the profile's connection setting	This is just a warning indicating that the setting in the profile is overruled by the scripted `OraFormsSetConnectMode`.
24	Profile setting INITIAL_ VERSION overridden by function `OraFormsSetInt`	This is just a warning indicating that the setting in the profile is overruled by the scripted `OraFormsSetInt ("INITIAL_ VERSION", version)`.
25	Verification succeeded	Whenever verification succeeds, a success log entry is created.
26	Handler already destroyed or not created yet	It is possible that the client applet sends messages from special handlers (controls) that are already officially destroyed by a destroy message from the server. This is not a real error and can only occur during recording. Therefore, a warning is logged.
27	Tree item was not found	All `OraFormsTree` API calls throw this error if the referenced tree item was not found in the tree.

Tips and Tricks

Explains issues that need to be considered when testing Oracle Forms applications with Silk Performer.

Oracle Forms Memory Usage

Oracle Forms Replay Engine uses Silk Performer’s Java Framework to send/receive Oracle Forms messages. Therefore each virtual user container (PerfRun) requires additional memory, as the Java Runtime must be hosted once per runtime process – not once per user executed in the runtime.

The initial footprint of the Java Runtime Environment depends on the JDK that is used for replay. The JDK must be defined in the profile settings of the Oracle Forms project. On average, 10 MB’s is required to host a Java Runtime. Additionally, each virtual user requires memory for Oracle Forms processing. The amount of required memory varies based on the following variables:

- Length of executed scripts
- Number of hosted controls
- Oracle Forms log level
- TrueLog On Error (enabled/disabled)

Significant memory is required for a log level that is higher than “Error.” “Normal” and “Debug” log levels log all Oracle Forms messages in the replay log and in the TrueLog’s In/Out Body section. This log information is helpful for error analysis, but requires significant additional overhead (the amount of memory required varies based on the length of scripts and the number of controls).

Latest Available JDK Version

The JDK needs to be configured in your profile settings. You should use the latest available JDK version for the following reasons:

- Performance: Newer versions perform better, allowing you to simulate more users on one machine.
- Stability: There might be stability issues when using older JDK versions, such as encountering deprecated functions. Those deprecated functions might result in unpredicted results during a load test.

There is no impact in using a different JDK version than the Oracle Forms client applet is using. During recording, the selected JDK is only used within the Silk Performer Recorder process, not by the applet. During replay it is used in the virtual user runtime.

Virtual Memory Size of a JVM

A Java Virtual Machine can be configured with minimum and maximum heap sizes. When the maximum heap size is reached, out of memory exceptions are thrown. During recording, a lot of memory is consumed. The amount varies based on the log level and the length of recorded transactions.

It is likely that you will run into a memory problem when using a default maximum heap size of only 16 MB. The Java runtime allows you to change the default values by defining two special runtime parameters:

- `-Xmsn` for defining the minimum heap size (for example: -Xms10m)
- `-Xmxn` for defining the maximum heap size (for example: -Xmx128m)

If you run into a memory problem during recording, please define the runtime parameter `-Xmx` with a high enough memory value in the runtime settings of your Oracle Forms applet viewer. In most cases, this can be set using the JInitiator configuration dialog.

Java VM JITter

When recording Oracle Forms 6i on either HTTP or Sockets, it is recommended that you disable Java JITting in the Java runtime settings. This is because the JITting library of the Java Virtual Machine (`symcjit.dll`) may crash your browser.

Note: Disable JITting whenever your browser crashes during recording.

Option #1

Define `-DJAVA.COMPILER=NONE` in the Java runtime settings of the Oracle **JInitiator Properties** dialog. This must be done using the JInitiator control panel, which is installed with JInitiator (available in the Windows Control Panel). To do this:

1. Launch the JInitiator **Control Panel**.
2. Add `-DJAVA.COMPILER=NONE` in the **Java Run Time Parameters** text field.
3. Click **Apply**.

Option #2

Use the check box on the Oracle **JInitiator Properties** dialog to disable the Just In Time Compiler (only available since version 1.1.8.x). To do this:

1. Launch the JInitiator **Control Panel**.
2. Select the **Advanced** tab.
3. Deselect the **Enable Just In Time Compiler** check box.
4. Click **Apply**.

Option #3

Define an environment variable: `SET JAVA_COMPILER=NONE`.

Recording Oracle Forms on Sockets

The communication channel between the Java applet on the client and the Oracle Forms server can either be via HTTP/HTTPS or raw TCP/IP sockets. To accurately record an Oracle Forms script when the application you want to record uses sockets, there is a setting that you must change in Silk Performer's **System Settings**.

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Click the **Proxies** tab.
4. Select the **SOCKS** proxy entry and click the **Edit** button. The **Proxy Settings** dialog appears.
5. In the **Suppress recording (only forward data)** field, enter the port that your Oracle Forms application uses for communication with the applet.
6. Click **OK**. If you fail to suppress the port, Silk Performer will record on both the Oracle Forms level and the TCP/IP level. This will generate a script with mixed `OraForms` and `WebTcp` calls.

Note: Disable the socks port of your application when your application uses raw socket communication.

Problems with Oracle Forms 6 Recording

When recording an Oracle Forms 6 application using the Oracle Forms 6 project type, Silk Performer listens on both transport channels: HTTP/HTTPS and Sockets.

Some applications that operate on the Socket layer have been known to crash Internet Explorer while both channels are being recorded. This is why Silk Performer allows you to specify the transport channel that is of interest to you.

This setting is defined in the **Recording Profile** dialog box.

1. In the Silk Performer menu, click **Settings** > **System** .
2. Click the **Recorder** icon. The **Recording Profiles** page opens.
3. Change the Java API to the protocol that you wish to record.

Now when you begin recording, do not select the preconfigured Oracle Forms 6i application type, use Internet Explorer as you have already changed its recording profile configuration.

Problems with Oracle Forms 6 Recording Using Additional Java Beans

Implementing Java Beans can extend Oracle Forms applications with new GUI-Elements or Client-Side functionality. Those Java extensions will be downloaded by the client applet at startup.

When recording Oracle Forms 6, Silk Performer's Recorder will hook the main Java Archive containing the applet classes. The Java extensions either derive from classes or implement an interface that is contained within this archive. The Java Runtime, depending on some configuration settings, will verify class signatures when loading those Java Beans. As the main archive has been modified, those verifications will fail and the applet will no longer start.

To disable this verification process it is necessary to define an additional Java runtime parameter in the **JInitiator Runtime Parameter** settings. Define the parameter `-noverify` in your JInitiator settings.

Oracle Applications 12i Support

Prerequisite for Recording

Important: Before recording Oracle Applications 12i you must disable ports 9000-9005.

Customized Recorded Scripts

Before you can replay a recorded Oracle Applications 12i script you need to customize the script.

The parameter icx_ticket contains session information for Oracle Applications 12i. The Silk Performer Recorder generates code to automatically parse the value of icx_ticket:

```
WebParseDataBoundEx(gsIcxTicket, STRING_COMPLETE, "icx_ticket='", 1, "'",
WEB_FLAG_IGNORE_WHITE_SPACE, 1);
WebPageParseUrl("IMG SRC", "IMG SRC=\"", "or", WEB_FLAG_IGNORE_WHITE_SPACE);
WebPageUrl("http://myserver.com:8005/OA_HTML/runforms.jsp", "Oracle
Applications R12",
OA_HTML_RUNFORMS_JSP003);
```

The `WebParseDataBoundEx` function tells the Silk Performer runtime to parse the value of `icx_ticket` from the next HTML page call into the `gsIcxTicket` variable.

In this example, the next HTML page call is `http://myserver.com:8005/OA_HTML/runforms.jsp`.

This page takes input parameters from the form `OA_HTML_RUNFORMS_JSP003`. If this form already contains a recorded value for `icx_ticket`, the value has to be set to an empty value.

```
OA_HTML_RUNFORMS_JSP003:
  "icx_ticket"                    := "", // hvg4LnVEVjLLFU-e1pNr6Q...
  "resp_app"                      := "PER",
  "resp_key"                      := "VU_HRMS_MANAGER",
  "secgrp_key"                    := "STANDARD",
  "start_func"                    := "PERWSGEB",
  "other_params"                  := "";
```

For all subsequent usages of the `icx_ticket` parameter, the value stored in the `icx_ticket` variable must be used instead of the recorded value.

Note:

The Silk Performer Recorder scripts customized code for the `OraFormsConnect` function by default. Verify that the customization has been performed correctly.

```
OraFormsConnect(
  "server module=/data/oracle/ebs/application/apps/apps_st/appl/fnd/12.0.0/
forms/US/FNDSCSGN
  fndnam=APPS record=names  config='VIS' icx_ticket='" + gsICXTicket + "'
resp='PER/VU_HRMS_MANAGER'
  secgrp='STANDARD' start_func='PERWSGEB' other_params=''", "http://myserver:
8005",
  "http://myserver.com:8005/forms/frmservlet?
&lookAndFeel=ORACLE&colorScheme=KHAKI&
serverApp=OracleApplications3&lang="US"&env=NLS_LANG='AMERICAN_AMERICA'+FORMS_
USER_DATE_FORMAT=
'DD-MON-RRRR'+FORMS_USER_DATETIME_FORMAT='DD-MON-RRRR+HH24%3AM"
"I
%3ASS'+NLS_DATE_LANGUAGE='AMERICAN'+NLS_SORT='BINARY'+NLS_NUMERIC_CHARACTERS='
.,'&
form_params=+config='VIS'+icx_ticket="'.hr7lGQhSIG3iAMWb9LkuVQ..'"+resp='PER
%2FVU_HRMS_MANAGER'+
secgrp='STANDARD'+start_func='PERWSGEB'+other_params=''&encoding="UTF-8");
```

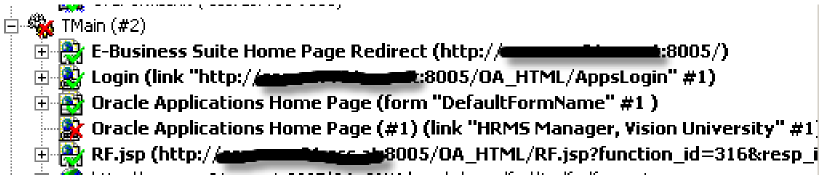
Customizing Oracle Applications 12i Session Information

Ensure that the `icx_ticket` parameter has been customized correctly.

1. Perform a trial run of the script by clicking the **Try Script** button on the **Workflow Bar**.

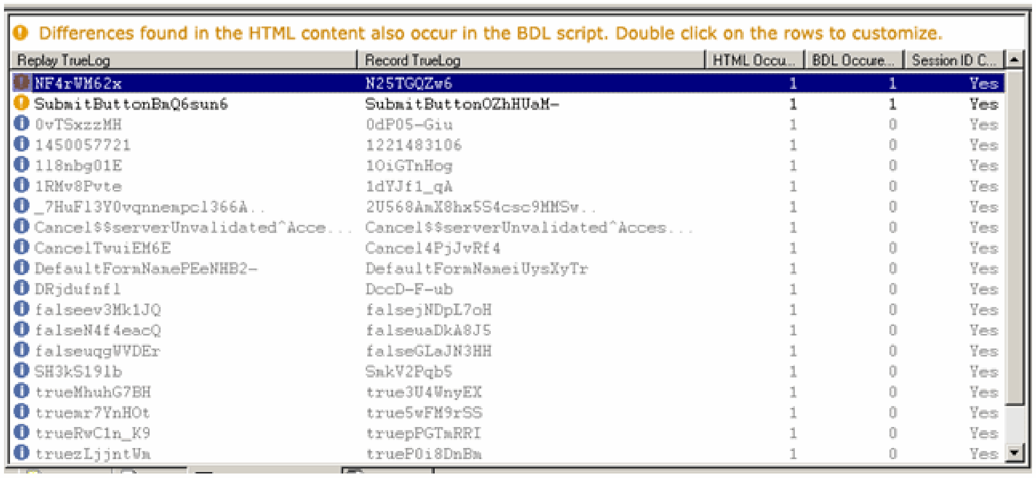
   Note: TrueLog generation must be enabled in Silk Performer

2. Open the resulting Try Script TrueLog with TrueLog Explorer. Typically, an HTML page (appearing soon after login) will show errors.
   Example error:



```
TMain (#2)
  E-Business Suite Home Page Redirect (http://[redacted]:8005/)
  Login (link "http://[redacted]:8005/OA_HTML/AppsLogin" #1)
  Oracle Applications Home Page (form "DefaultFormName" #1 )
  Oracle Applications Home Page (#1) (link "HRMS Manager, Vision University" #1
  RF.jsp (http://[redacted]8005/OA_HTML/RF.jsp?function_id=316&resp_i
```

3. Click the **Analyze Test** button on the **Workflow Bar**.
4. Select **Compare your test run** on the **Workflow - Analyze Test** wizard page.
5. Close the **Step through Truelog** dialog.
6. Select the TrueLog node that corresponds to the recorded application's login page.
7. Right-click the node and select **Synchronize Truelogs** from the context menu.
8. Select **Edit** > **Truelog Type** > **Web** from the TrueLog Explorer menu bar.
9. Click the **Differences** tab.



Differences found in the HTML content also occur in the BDL script. Double click on the rows to customize.

Replay TrueLog	Record TrueLog	HTML Occu...	BDL Occure...	Session ID C...
NF4rWM62x	N25TGQZw6	1	1	Yes
SubmitButtonBmQ6sun6	SubmitButtonOZhHUaM-	1	1	Yes
0vTSxzzMH	0dP05-Giu	1	0	Yes
1450057721	1221483106	1	0	Yes
118nbg01E	10iGTnHog	1	0	Yes
1RMv8Pvte	1dYJf1_qA	1	0	Yes
_7HuF13Y0vqnnempcl366A..	2U568AmX8hx5S4csc9MMSw..	1	0	Yes
Cancel$$serverUnvalidated^Acce...	Cancel$$serverUnvalidated^Acces...	1	0	Yes
CancelTwuiEM6E	Cancel4PjJvRf4	1	0	Yes
DefaultFormNamePEeNHB2-	DefaultFormNameiUysXyTr	1	0	Yes
DRjdufnfl	DccD-F-ub	1	0	Yes
falseev3Mk1JQ	falsejNDpL7oH	1	0	Yes
falseN4f4eacQ	falseuaDkA8J5	1	0	Yes
falseuqgWVDEr	falseGLaJN3HH	1	0	Yes
SH3kS191b	SmkV2Pqb5	1	0	Yes
trueMhuhG7BH	true3U4WnyEX	1	0	Yes
truemr7YnHOt	true5wFM9rSS	1	0	Yes
trueRwC1n_K9	truepPGTmRRI	1	0	Yes
truezLjjntWm	trueP0i8DnBm	1	0	Yes

10. Right-click any rows that show differences between the recorded TrueLog and the replay TrueLog that also occur in the script and therefore likely contain session information (highlighted in yellow) that should be customized.
11. Run another Try Script run in Silk Performer to verify that your session information customizations have been successful.

Adding a New Java Bean to a System Classpath

Oracle Forms applications can be pushed using Java Beans components. Java Beans, in this context, are Java archives that contain applications that can be run inside Oracle Forms applets. Examples of such applications are calendars and clocks.

There are requirements for testing Oracle Forms applications that make use of Java Beans. Ensure that the Java archives containing the Java Beans are also in the system classpath. To add a new Java Bean to a system classpath, follow these steps:

1. Navigate to **Start** > **Control Panel** > **System**.
2. Select the **Advanced** tab.
3. Click **Environment Variables**. This launches the **Environment Variables** dialog box where you can edit the environment variables.
4. Select the *CLASSPATH* variable in the **System variables** box and click **Edit**.
5. In the **Edit System Variable** dialog box, update the **Variable name** and **Variable value** to add the Java archive that contains the Java Bean.
6. Click **OK**.
7. Restart the Oracle Forms server.

SAP Support

This section explains Silk Performer support for testing of SAP applications, including project setup and SAP configuration. It also provides tips for customizing scripts and including verification functions.

SAP eCATT Support

SAP eCATT has been integrated with Silk Performer. SAP's eCATT facility allows you to create test scripts in SAP using the scripting language of your choice. eCATT allows you to use external test tools (for example, Silk Performer) while utilizing eCATT as a repository for your test scripts. eCATT also serves as a basic test management solution for triggering script executions. Not only can both internal and external scripts be executed individually, they can also be combined and executed in sequence.

eCATT offers import arguments, a mechanism for calling scripts with special input values. Scripts can not only receive input values, scripts can also set output values when they are executed; scripts can be executed in sequence, using input values derived from the output values of earlier script executions.

For full details regarding Silk Performer's integration with SAP eCATT, refer to the SAPGUI Tutorial. For more information regarding eCATT, consult the SAP documentation.

Configuring a SAP eCATT Connection

Connection details for Silk Performer's communication with SAP eCATT must be specified in Silk Performer system settings. There are two options for connecting to SAP: You can either specify a `SAPLOGONID` or you can specify `AS Host`, `RFC Type`, and `SystemNr` settings. With either option you must specify client, language, username, and password details. Note that when you select a `SAPLOGONID`, the `AS Host`, `RFC Type`, and `SystemNr` fields are grayed out.

1. Navigate to **Settings** > **System** .
2. On the **System Settings – Workbench** dialog, select the **SAPGUI** group icon.
3. The **eCATT Connection** page is selected by default. From the **SAPLogon** list box, select your SAP login ID.

   Note: This box is preconfigured with all available SAP login IDs.
4. In the **AS Host** field, enter the combined router/application-server string (for example, `H/195.61.176.22/H/194.117.106.130/S/3297/H/cpce801`).
5. In the **RFC Type** field, enter either 3 (for R/3) or 2 (for R/2).
6. In the **System NR** field, enter the SAP system number.
7. In the **Client** field, enter the internal client ID number from the SAP server (this is the value that must be entered on the SAP login screen).

8. From the **Language** list box, select your language preference. The values `EN` (English) and `DE` (German) are preconfigured, though you can specify any other language abbreviation string if that language is installed on the SAP system.
9. In the **Username** field, enter your SAP eCATT username.
10. In the **Password** field, enter your SAP eCATT password.
11. Once you have completed this dialog, click **Test Connection** to confirm that you have specified accurate connection details.
12. Click **OK** to save your settings.

Configuring eCATT Extended Results

To enable the viewing of Silk Performer result files from within SAP GUI, you can specify a UNC path to a public file share in which extended Silk Performer test results can be stored and accessed by users (for example, `\\fileserver\ecattresults`). Silk Performer will use the specified directory to store the results of Silk Performer test executions initiated via SAP eCATT. Users can easily access test results by clicking a link in the SAP eCATT GUI.

1. Navigate to **Settings** > **System** .
2. On the **System Settings – Workbench** dialog, select the **SAPGUI** group icon.
3. Click the **eCATT Results** tab.
4. Check the **Use SAP extended results** check box.
5. In the **SAP extended results root directory** field, browse to and select the directory that is to be used for SAP extended results.
6. Click **OK** to save your settings.

SAPGUI Support

This section provides guidelines for customizing scripts (including verification functions) so that they can be run in distributed multi-user environments. For detailed instructions on how to use Silk Performer to test SAP systems via the SAPGUI interface, refer to the SAPGUI Tutorial.

Basic Concepts

SAP has a test interface that can be used by tool vendors to test SAP systems. This test interface offers the ability to listen to events as well as to execute user actions. Silk Performer uses this interface to record new scripts and for script execution.

This section provides an overview of how to use Silk Performer to test SAP systems via the SAPGUI interface and outlines possible server configurations and discusses how they can be tested.

SAPGUI Scripting

SAPGUI scripting must be installed on the client. It can be selected during the SAPGUI client setup process. Once it has been installed on the client, SAPGUI scripting must be enabled on both the client and the server.

Note: The SAP security parameter `sapgui/user_scripting_disable_recording` blocks access to key events on the SAP server that Silk Performer needs access to in order to facilitate replay and recording. Enabling the parameter `sapgui/user_scripting_disable_recording` results in replay failure. To allow replay, you must disable this parameter.

During recording

The SAPLogon process initializes the SAPGUI scripting API. This allows the recorder to attach to a COM object that offers full access to the SAP DOM where each component (control) within SAPGUI can be accessed. This COM interface also offers an event mechanism that notifies subscribers of changes in the GUI (for example, button presses, text edits) and server roundtrips (Start/End Request).

The Silk Performer Recorder is a subscriber of this event interface, so it receives notice of all changes that are made by the user in the SAPGUI user interface. Depending on recording profile settings, Silk Performer Recorder is either set for low-level scripting of API calls for each change event, or high-level scripting for API calls in which certain change events are merged with high-level calls (for example, `SapGuiLogon`).

During replay

When executing a SAPGUI load test, the SAPGUI user interface is automatically switched to invisible mode. There is a setting in the profile settings that allows you to execute a GUI-less TryScript run. This setting however, has no effect in load test scenarios as it does not make sense to show the UI when executing multiple users on a machine, because too many windows would be displayed; additional resource consumption would be required if each window has to render itself.

As each VUser has access to the full SAP DOM, it is possible to generate TrueLogs with rich control information.

SAP Patch Levels

Silk Performer's SAPGUI record/replay technology is based on the SAPGUI scripting API, which needs to be enabled on the server and client side.

SAPGUI scripting API is not available in all versions of SAPGUI client – you should therefore check your Patch Level.

Patch Levels on the Server

The components on the server also require a certain patch level. Following is an overview about the required patch levels:

Software Component	Release	Package Name	Kernel Patch Level
SAP_APPL	31I	SAPKH31I96	Kernel 3.1I level 650
SAP_APPL	40B	SAPKH40B71	Kernel 4.0B level 903
SAP_APPL	45B	SAPKH45B49	Kernel 4.5B level 753
SAP_BASIS	46B	SAPKB46B37	Kernel 4.6D level 948
SAP_BASIS	46C	SAPKB46C29	Kernel 4.6D level 948
SAP_BASIS	46D	SAPKB46D17	Kernel 4.6D level 948
SAP_BASIS	610	SAPKB61012	Kernel 6.10 level 360

Note: On the client side, SAPGUI client 620 is required with a minimum patch level of 44.

Checking the SAP Patch Level

SAPGUI scripting is not supported by all versions of SAP. Therefore it is necessary to check if your installation offers this kind of support. You can do this by checking your current Patch Level. The Patch Level needs to be at least 66.

1. Start your SAPGUI logon window which is normally accessible through **Start** > **Programs** > **SAP Front End** > **SAPLogon** > **About SAP Logon**. The **SAP Version Information** dialog box appears.
2. Check that your Patch Level is at least 66.

Checking the SAP Patch Level on the Server

1. Logon to your SAP system.
2. Select **System** > **Status**.

3. Click the **Other Kernel Information** button. The **Kernel Information** dialog opens.
4. Check the value of the field **Sup. Pkg. lvl**.

   Check your value and Kernel Release version with the table in SAP Patch Levels.

   If your Patch Version is lower than the required one update your system. Refer to the SAP OSS note #480149 for detailed instructions.

Checking SAP R/3 Support Packages

1. Logon to your SAP system.
2. Run the SPAM transaction.
3. In the **Directory** section, select **All Support Packages**, and click the **Display** button.
4. If you have SAP_BASIS, 4.6C installed, verify that SAPKB46C29 is installed.

   You will see a green circle if it is installed. If you do not have the OCS package installed, download it from *www.sap.com* and install it. For more information, refer to the SAP OSS note #480149.

Enabling SAPGUI Scripting on the Server

Silk Performer's SAPGUI record/replay technology is based on the SAPGUI scripting API, which needs to be enabled on the server and client side. SAP servers of version 620 and higher offer SAPGUI scripting support. Earlier versions require certain patch levels.

A user with administrative privileges needs to change the `sapgui/user_scripting` parameter in order to enable scripting. The value needs to be changed to `TRUE`.

1. Logon to your SAP server.
2. Run transaction `RZ11`. Specify the parameter name `sapgui/user_scripting` and click **Display**.

   If `Parameter name is unknown` appears in the status bar, this indicates that you are missing the current support package. Check your installed packages.
3. Change the value to `TRUE`.
4. Click **Save**.
5. Restart the application server, since this change only takes effect when you log onto the system.

Enabling SAPGUI Scripting API on the Client

All clients that are either used for recording, replay or monitoring must have a SAPGUI client with SAPGUI scripting installed. You have to select this option in the SAPGUI Setup.

After the installation you need to make sure that SAPGUI scripting is enabled:

1. Start your SAPLogon and go to the logon screen of one of your servers.
2. Open the **Options** dialog in your SAP client.
3. Select the **Scripting** tab, check **Enable Scripting**, and disable both sub options. This prevents warning messages when Silk Performer interacts with SAPGUI scripting GUI.

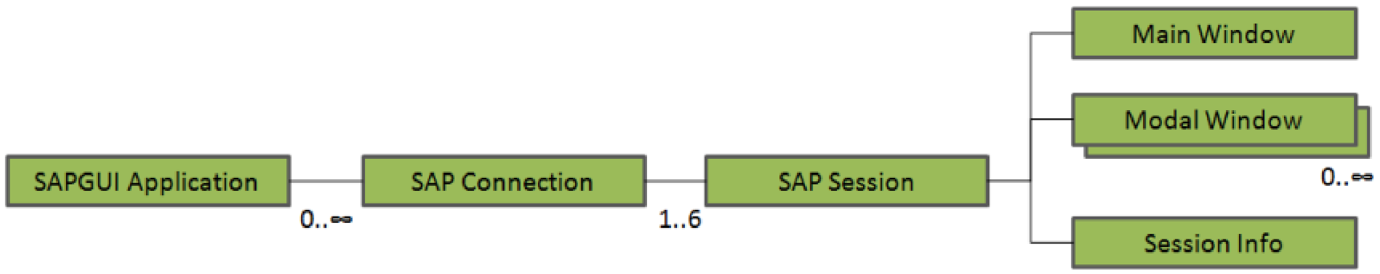
If the installation status of the **Scripting** tab does not display `Scripting is installed`, you must install SAPGUI scripting by choosing the option in the SAPGUI client setup process.

SAPGUI Application Architecture

The SAPGUI application object, which is instantiated once per virtual user in test scenarios, can have infinite SAP connections. A connection can be opened with either `SapGuiOpenConnection` or `SapGuiOpenConnectionByName`. One connection should be opened by each virtual user that you want to simulate at the beginning of your test. When opening a new connection, an SAP session is created as well. A connection can hold up to 6 sessions, which can be created with `SapGuiCreateSession`.

When the last session on a connection is closed, the connection itself is closed and destroyed.

An SAP session holds the information about all control elements on the main window of the session. Additionally, a session info object can be accessed for specific data about the current transaction or performance related data.



An SAP session usually has a main window as its child. This window is the starting point for all user interaction with the current session.

Object IDs

Each object within a SAPGUI application is identified by a unique ID, which contains the ID information of the complete object hierarchy, beginning with the application object itself. The IDs of the objects in the hierarchy are separated by forward slashes ("/") where the application object itself begins with a forward slash.

Object	ID Description
SAP Application	The application's ID is `/app`.
SAP Connection	A connection's partial ID is `con[X]`, where X is the index of the connection. The index is required when there are multiple objects with the same base ID. The index is 0-based. Example: `/app/con[0]`
SAP Session	The session's partial ID is `ses[X]`, where X is the index of the session within the current connection. Example: `/app/con[0]/ses[0]`
SAP Window or Dialog	The window's partial ID is `wnd[X]`, where X is the index of the window within the current session. Example: `/app/con[0]/ses[0]/wnd[0]`
SAP User Area	The ID of the user area is `usr`. As there is only one user area per window, there is no need for an index here. Example: `/app/con[0]/ses[0]/wnd[0]/usr`
SAP Title Bar	The ID of the title bar is `titl`. It is a child of a window or dialog. Example: `/app/con[0]/ses[0]/wnd[0]/titl`
SAP Tool Bar	The partial ID is `tbar[X]`. There can be up to two tool bars on a window and therefore the index is needed.

Object	ID Description
	Example: `/app/con[0]/ses[0]/wnd[0]/tbar[0]`
SAP Status Bar	The ID of the status bar is `sbar`. As there is only one status bar per window, there is no need for an index here. Example: `/app/con[0]/ses[0]/wnd[0]/sbar`

Other Controls	Certain control types have special ID prefixes (for example, `lbl` for labels and `txt` for text controls). Following the prefix, controls have a programmatical name (for example, `RSYST-BCODE`).
	Controls that are listed in a table format can have multi-dimensional indices, similar to the indices shown above for sessions, connections, etc.
	A multi-dimensional index is defined as `controlid[X,Y]`.
	Examples:
	`/app/con[0]/ses[0]/wnd[0]/usr/txtRSYST-BCODE`
	`/app/con[0]/ses[0]/wnd[0]/usr/sub:SAPMSYST:0020/txtINFO-TABTDLINE[0,0]`
	`/app/con[0]/ses[0]/wnd[0]/tbar[0]/btn[0]`

Object Access with Silk Performer

To make object access easier, Silk Performer API calls do not require full object IDs for controls. Instead, the relative object ID of the current window can be used for most API calls.

For example, pressing the button `/app/con[0]/ses[0]/wnd[0]/tbar[0]/btn[0]` can be done with `SapGuiPressButton ("tbar[0]/btn[0]");`.

To do this, Silk Performer requires the information about the current window, session, and connection. For this purpose there are three methods that allow you to set the current context:

- `SapGuiSetActiveConnection`
- `SapGuiSetActiveSession`
- `SapGuiSetActiveWindow`

When you record a script you see one of the three above mentioned functions scripted wherever the context has changed.

A typical script looks like this:

```
gsConnID := SapGuiOpenConnection("CONNECTSTRING");
SapGuiSetActiveConnection(gsConnID);
SapGuiSetActiveSession("ses[0]");
SapGuiSetActiveWindow("wnd[0]");
SapGuiPressButton("/tbar[0]/btn[0]");
```

This makes scripts easier to read and customize.

Accessing Object Properties

Each object has properties that can be accessed from a Silk Performer script during execution. To get a full list of all properties of each object type, do the following:

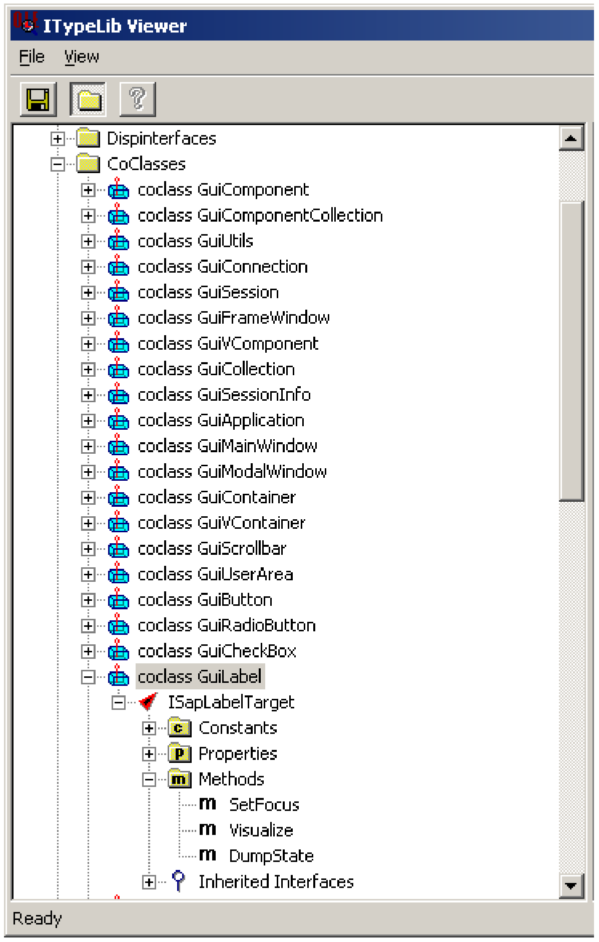
1. Open a tool that allows you to view type libraries (for example, OLEView, which comes with Microsoft Visual Studio).
2. Open the file `<SAPINSTALLDIR>\FrontEnd\SAPGui\sapfewse.ocx`. `<SAPINSTALLDIR>` is usually located in `C:\Program Files\SAP`.
3. Explore the different COM classes for their properties. COM classes have meaningful names so that it is easy to find the correct COM objects for your requested object type, for example `GUITextField` for text controls.
4. Special controls such as trees and grids can be explored by opening the corresponding `dll`, located in `<SAPINSTALLDIR>\FrontEnd\Controls\Scripting`. For example, open `GridViewScripting.dll` to get the information for grid controls.

Silk Performer offers methods to `Get/Set` properties, and even for invoking methods. If a method call or a `GetProperty` access returns another object, the object can be accessed with additional `Get/Set/Invoke` calls by passing the constant `SAPGUI_ACTIVEOBJECT` as the object identifier.

Accessing Low-Level Properties

The functions `SapGuiInvokeMethod`, `SapGuiSetProperty`, and `SapGuiGetProperty` can be used to access the low-level properties of any control on the active screen. This makes it possible to, for example, confirm that a text control is read-only, or to determine the background color of a label.

The SAPGUI scripting API that is used for SAPGUI testing is a large COM library that allows Silk Performer to access controls and perform actions. This same COM library can be used with the above mentioned API calls. To access the list of methods and properties that individual controls offer, you must inspect the type library of the SAPGUI scripting API. For this you need a tool that facilitates the inspection of type libraries (for example the Ole32View tool that comes with Microsoft Visual Studio). With the tool, open the `sapfewse.ocx` file, which can be found in the SAPGUI installation directory under `\FrontEnd\SapGui`. The image below shows the properties that can be accessed on a label control:



To get the name of a control for which you know the control's ID, use the following call:

```
SapGuiGetProperty("/usr/lbl[1,2]", "Name", sOutValue);
Print("The control has the following name:" + sOutValue);
```

Most properties return a simple data type such as a string, number, or boolean parameter. Some properties return other objects, for example the `Parent` property returns the parent control of the current control. When a property returns another control, the control is temporarily cached and can be accessed via the constant `SAPGUI_ACTIVEOBJECT`. Here is a code example for retrieving the name of a parent property:

```
SapGuiGetProperty("/usr/lbl[1,2]", "Parent");
SapGuiGetProperty(SAPGUI_ACTIVEOBJECT, "Name", sOutValue);
Print("The parent control has the following name:" + sOutValue);
```

Differences Between 620 and 640 Clients

SAP introduced new properties and methods with its SAPGUI 640 client. With 640 you can, for example, get the background color of a label or checkbox. So, when you develop a script on 640 you must ensure that your agents also run on 640 and that the same patch level is used. Otherwise you may request properties that are not available on the agent and generate an error.

Property Overview

SAP is comprised of components that have the following properties (ComClass GuiComponent):

- 'Name' - Name of the control
- 'Type' - The control type in text format (for example GuiButton, GuiTextField)
- 'TypeAsNumber' - All types have internal numbers (for example 30=GuiLabel, 31=GuiTextField)

- 'ContainerType' - A Boolean property that defines whether or not a control is a container. Containers contain other controls as children (for example a toolbar is a container that contains toolbar buttons).
- 'Id' - The unique ID of a control
- 'Parent' - When a control is contained in a container this property returns the parent control

Visual components, such as controls, have additional properties (ComClass GuiVComponent):

- 'Text' - The main text of a control (for example the text in a text control or the text on a button)
- 'Left', 'Top', 'Width', 'Height', 'ScreenLeft', 'ScreenTop' - Number values offering details about screen coordinates and coordinates within a parent container
- 'Changeable', 'Modified' - Boolean parameters that indicate the current state of a control (is the control changeable or read-only; has the control been modified, etc)

Each control type may have additional properties which can be seen in the COM type library via your type library inspection tool.

Installation and Requirements

To use Silk Performer's support for SAPGUI, SAPGUI scripting must be installed and enabled on the client and the server side.

Enabling SAPGUI Scripting on the Server

Silk Performer’s SAPGUI record/replay technology is based on the SAPGUI scripting API, which needs to be enabled on the server and client side. SAP servers of version 620 and higher offer SAPGUI scripting support. Earlier versions require certain patch levels.

A user with administrative privileges needs to change the `sapgui/user_scripting` parameter in order to enable scripting. The value needs to be changed to `TRUE`.

1. Logon to your SAP server.
2. Run transaction `RZ11`. Specify the parameter name `sapgui/user_scripting` and click **Display**.

   If `Parameter name is unknown` appears in the status bar, this indicates that you are missing the current support package. Check your installed packages.
3. Change the value to `TRUE`.
4. Click **Save**.
5. Restart the application server, since this change only takes effect when you log onto the system.

Enabling SAPGUI Scripting API on the Client

All clients that are either used for recording, replay or monitoring must have a SAPGUI client with SAPGUI scripting installed. You have to select this option in the SAPGUI Setup.

After the installation you need to make sure that SAPGUI scripting is enabled:

1. Start your SAPLogon and go to the logon screen of one of your servers.
2. Open the **Options** dialog in your SAP client.
3. Select the **Scripting** tab, check **Enable Scripting**, and disable both sub options. This prevents warning messages when Silk Performer interacts with SAPGUI scripting GUI.

If the installation status of the **Scripting** tab does not display `Scripting is installed`, you must install SAPGUI scripting by choosing the option in the SAPGUI client setup process.

Recording and Replay Concepts

The following sections explain the workflow for recording, replaying and customizing scripts, as well as result analysis.

Recording Settings

Settings specific for recording can be changed on the recording tab of your profile settings:

- **Script logon as single function** - If enabled, the logon procedure will be scripted as `SapGuiLogon` API call. If disabled, multiple API calls like setting username, password and pressing `ENTER` will be scripted.
- **Script low level functions** - Instead of high level API functions, like for example `SapGuiSetText`, low level API functions will be scripted, such as `SapGuiInvokeMethod`, `SapGuiSetProperty`, …
- **Script timers** - Most of the SAPGUI API functions take an optional timer parameter. If this parameter is defined, measures for this will be generated during replay. If this option is enabled, SAPGUI Recorder will automatically script appropriate timer names for each function.
- **Attach to existing SAP session** - If enabled, SAPGUI Recorder will attach to an existing SAPGUI session without recording the `SapGuiOpenConnection` statement.
- **Record window title verification** - If enabled, SAPGUI Recorder will script the `SapGuiSetActiveWindow` with the window title in order to be verified during replay.

Common Settings

The following settings are common settings for both recording and replaying:

- **Log level** - Defines the level of logging. For troubleshooting, use `Debug`, otherwise use `Normal`. When running large load tests logging should be `Disabled` in order to reduce memory consumption.
- **Capture screenshots** - If enabled, screenshots will be captured for every new window that is activated. This option is only available if **Show SAPGUI during single runs** is enabled when replaying the script.
- **Capture screenshots for every action** - If enabled, screenshots will be captured for every user action that causes a roundtrip to the SAP server. This option is only available if **Capture screenshots** is enabled.
- **Log control information in TrueLog** - If enabled, control information about every control on the active window will be logged to the TrueLog. This allows you to use the customization feature in TrueLog Explorer. This option should be disabled when running load tests as it implies additional resource consumption.
- **Log control information for single run in TrueLog** - Same as **Log control information in TrueLog**, but only for try script and verification runs.
- **Log control information on error** - If enabled, control information about every control on the active window will be logged to the TrueLog in case of an error during replay. This allows you to troubleshoot your replay problem by getting the current state of all controls on the screen where the error happened. It is recommended to use this option during a load test rather than **Log control information in TrueLog**.

Default Profile Settings

When recording an SAP application using the default profile settings (capture screenshots and log control information), the SAP client may appear slower than usual. This is due to the extra overhead required for taking screenshots and logging control information for all controls on each SAP window. This option should not be changed however as you will lose the many benefits that the TrueLog provides. However if you experience timing problems, switch off the options in this order:

- **Capture screenshots for every action**
- **Capture screenshots**
- **Log control information in TrueLog**

Replay Settings

Settings specific for replaying can be changed on the replay tab of your profile settings:

- **Replay timeout** - Defines the timeout during replay. If there is no response from the server within this timeout period an error will be thrown.
- **Show SAPGUI during single runs** - If enabled, the SAPGUI client will be shown during replay. This option should only be used for a Try-Script run. During a load test it should be turned of.
- **Enable client-side scripting** - SAPGUI scripting needs to be enabled on every client machine via the **Options** menu in the SAPGUI client application. When running a test on multiple agents it would imply that this setting need to be changed manually on every machine before starting the load test. By enabling this option, Silk Performer is changing the setting automatically on every agent before starting the load test.
- **Use new SAP Visual Design** - SAPGUI can be run in two visual modes, the old or the new visual design mode. The setting normally needs to be changed with the SAP Configuration Tool. By enabling/ disabling this option, Silk Performer does this change automatically before starting the load test. This option allows you to test the different behavior in terms of resource consumption when using either the old or new visual design.

Timers and Counters

The **Measuring** tab contains settings for replay measuring. You can either **Enable all timers and counters for all controls**, or select those that are of interest to you. Timers will only be created for method calls that have the optional timer parameter specified.

For a description about those timers, see SAP Results.

Password for SapGuiLogon

Values in password fields cannot be retrieved via the SAPGUI scripting API. Silk Performer's Recorder therefore is not able to script values entered into password fields as it can on the logon screen.

`SapGuiLogon` is generated with asterisks in the password parameter. You must customize this value manually in the script or by using TrueLog Explorer after a TryScript run.

When running a TryScript or even a load test with an uncustomized password field of the `SapGuiLogon` method, the actual logon will not be executed. Instead, an error of severity `process exit` is thrown, because if you run a load test in which you forget to customize the password parameter, the user will be locked out after 3 attempts. To prevent this, Silk Performer stops script execution when the password parameter is not customized.

Setting SAPGUI Options

Silk Performer offers a tutorial that walks you through the process of testing SAPGUI applications. The tutorial is available at **Start** > **All Programs** > **Silk** > **Silk Performer 20.0** > **Documentation** > **Tutorials** > **SAP** .

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **SAPGUI** icon. On the **General** page you can define options for how the SAP Recorder records and replays scripts.
5. Login to SAP systems involves several user interactions. With the **Script logon as a single function** option enabled, the recorder only scripts one function that performs all the required login actions.
6. Check the **Script low level functions** check box to use a basic set of scripting functions instead of object-specific functions.

7. Check the **Script timers** check box to script timers automatically.

   These timers are used to measure SAP API calls.
8. Check the **Attach to existing SAP session** check box to have the recorder attach to an existing session rather than create a new session.
9. Check the **Record window title verification** check box to record `SapGuiSetActiveWindow` functions in such a way that title verifications are automatically performed on activated windows during test runs.
10. In the shortcut list on the left, click the **Replay** button. The Replay category is displayed.
11. In the **Replay timeout** text box, specify a timeout period (in seconds) for automatic shutdown of `SAPFewgsrv` (when not otherwise closed at the end of test runs).
12. Check the **Show SAPGUI during single runs** check box to display the R/3 client during test runs.

    GUI display is only an option for Try Script executions. This setting is ignored for baseline tests and load tests.
13. Check the **Enable client-side scripting** check box to enable SAP client-side scripting via the registry.

    This option disables warnings that raise pop-up windows when new users start.
14. Check the **Use new SAP Visual Design** check box to enable SAP Visual Design for the SAP client.
15. Click the **Logging** tab to specify additional log levels.

    Note: These settings affect performance and resource utilization.
16. From the **Log level** list box, select a log level for virtual user logging:
    - `Disable` - Virtual user logging is disabled.
    - `Normal` - Default logging.
    - `Debug` - Detailed information is logged, such as additional session information and errors.
17. Check the **Capture screenshots** check box to generate screenshots for each activated window.

    Note: Screenshot capture is only enabled for Try Script executions. This setting is ignored for baseline tests and load tests.
18. Check the **Capture screenshots for every action** check box to have each action performed by the script documented with a screenshot.

    Note: Screenshot capture is only enabled for Try Script executions. This setting is ignored for baseline tests and load tests.
19. Check the **Log control information in TrueLog** check box to have additional control information written to the TrueLog.
20. Check the **Log control information for single run in TrueLog** check box to have additional control information written to the TrueLog only for try script and verification tests.
21. Check the **Log control information on error** check box to enable writing of control information to TrueLogs in the case of errors.

    Note:

    For large load-tests, it is also recommended that you uncheck **Log control information in TrueLog** and only select **Log control information on error**. These settings greatly improve overall performance, as not all information on each screen is logged. In the case of errors, control state is logged on the nodes that cause the errors; this allows you to troubleshoot problems, as you can see the full state of the active window.
22. Check the **Highlight controls** check box to have controls highlighted as they are accessed by the script.
23. Click the **Measuring** tab to specify a custom measurement level.

    By default, all available performance metrics are collected during test runs.
24. Check the **Enable all timers and counters for all controls** check box to enable all timers and counters for all actions on controls during replay.Or, select a specific control from the **Control** list box and select the specific counter types that you want to have applied to that control:

- `Count round-trips` - The number of round-trips for the action on the control.
- `Count response time` - The time it takes a function to return requested data.
- `Count Interpretation time` - The time it takes to interpret a request.
- `Count flushes` - The number of flushes per action.

Note: Alternatively, you can click the **Apply to All Controls** button to apply your counter settings to all controls.

25. Click **OK** to save your settings.

Files Generated During Recording

Script

The script contains the `SapGuiOpenConnection` statement as the first statement in the `TMain` transaction, unless you have not selected the **Attach to existing SAP Session** option.

The complete logon sequence, from `SapGuiOpenConnection` to `SapGuiLogon`, should be moved to the `Tinit` transaction when performing large load tests.

When high-level scripting is enabled, the recorder scripts high-level API functions for most control interactions. For all others (where there are no high-level API calls) the low-level functions `SapGuiInvokeMethod` and `SapGuiSetProperty` are scripted.

Recording Text Logs

Text logs contain detailed information about SAPGUI events and controls when the log level is set to `Debug`. The additional information in this log is helpful for troubleshooting. If you experience a problem and need to contact technical support, switch the log level to `Debug` before recording. This also applies to the replay text log file.

Record TrueLogs

Record TrueLogs contain the same information as replay TrueLogs. This allows you to use TrueLog Explorer's Compare view after you execute a TryScript.

SAP Results

Depending on the measure settings in the active profile, measures are generated for those method calls that have the optional timer parameter defined.

Measures are generated for those method calls that force a roundtrip to the SAP server.

Note: Not all API calls force a roundtrip.

The following measures will be generated during a roundtrip:

Round Trips

Before SAPGUI sends data to the server, it locks the user interface. In many cases it will not unlock the interface once data arrives from the server, but instead will send a new request to the server immediately. Controls in particular use this technology to load the data they need for visualization. The count of these token switches between SAPGUI and the server is offered with this measure.

Flushes

Counts the number of flushes in the automation queue during server communication.

Interpretation Times

The interpretation time begins after the data has arrived from the server. It comprises the parsing of the data and distribution to the SAPGUI elements.

Response Times

This is the time that is spent on network communication from the moment data is sent to the server to the moment the server response arrives.

An overall counter for the roundtrips is shown in the Monitor window during a test and can also be monitored in Performance Explorer as one of the Silk Performer Controller/Agent measures.

SAP Performance Monitoring

Use Performance Explorer to monitor performance and reliability metrics of a SAP system. Performance Explorer is shipped with two monitors specially designed for monitoring SAP installations.

1. On the **Real-Time Monitoring** tab, in the **Monitor** group, click **System**. The **Data Source** wizard opens.
2. Click the **Select from predefined Data Sources** option button.
3. Expand the **Application Server** folder and the **SAP** folder.
4. Depending on the type of SAPGUI monitor that you want to use, select the appropriate monitor type.

   The following types of SAPGUI monitors are available:

   - SAPGUI Monitoring (ST02): Monitors buffer related metrics. SAP transaction `ST02` is executed when running this monitor.
   - SAPGUI Monitoring (ST03N): Monitors application specific metrics. SAP transaction `´ST03n` is executed when running this monitor.
   - SAPGUI Monitoring (ST04): Monitors database related metrics. SAP transaction `ST04` is executed when running this monitor.
   - SAPGUI Monitoring (ST07): Monitors user distribution on the SAP system. SAP transaction `ST07` is executed when running this monitor.
   - SAPGUI OS-Monitoring (ST06): Monitors operating system specific metrics. SAP transaction `ST06` is executed when running this monitor.
5. Click **Next**.

   The **Connection parameters** page opens.
6. In the **Hostname** text box, specify the host of the machine to be monitored

   This value is for display purposes only. It is not used in the monitor itself.
7. Click **Next**. The **Attributes Configuration** page opens.
8. Define the following monitoring-specific attributes:

   - **ConnectionString** – Complete connection string to the SAP server. If you are not sure, record the logon sequence with Silk Performer. The connection string is the first parameter of `SapGuiOpenConnection`.
   - **Username** – SAP username with access rights to the monitoring transaction.
   - **Password** – Password for the SAP user.
   - **ClientNum** – Client number for the logon procedure, such as `850`.
   - **Language** – Language to use for logon, such as `EN`.
   - **Entity** – The data entity that should be monitored, such as `Dialog`, `RFC`, or `Background`.

     Note: The **Entity** attribute does not have to be provided for the SAP OS-Monitor.
   - **TimeFrame** – SAP provides the average values of the past interval that you define.

Note: The **Timeframe** attribute does not have to be provided for the SAP OS-Monitor.

- **Server** - *(ST03N only)*The server that should be monitored. Its the name of the tree node that is to be selected in ST03N. Total will return measures from the overall SAP System

9. Click **OK**. The **Select displayed measures** page opens.
10. Check the check boxes for those measures that you want to include in the initial monitor view and then click **Finish**.

A connection to the specified host is established, and an initial view that contains the measures you selected is displayed.

SAPGUI TrueLogs

TrueLog Explorer is a powerful tool that not only offers a convenient means of exploring SAPGUI applications, it also enables you to add verifications for control values and customize input values for controls that are modified during recording.

SAPGUI TrueLog Structure

SAPGUI TrueLogs have a similar structure to Oracle Forms TrueLogs. Each `SapGuiSetActiveWindow` call results in a new top-level node. A `SapGuiSetActiveWindow` call is scripted for each window that is activated during a recording session. All actions that are performed on windows (for example, control edits, list entry selections) are shown as sub-nodes grouped by a virtual `SapGuiRoundtrip` node.

Node Information

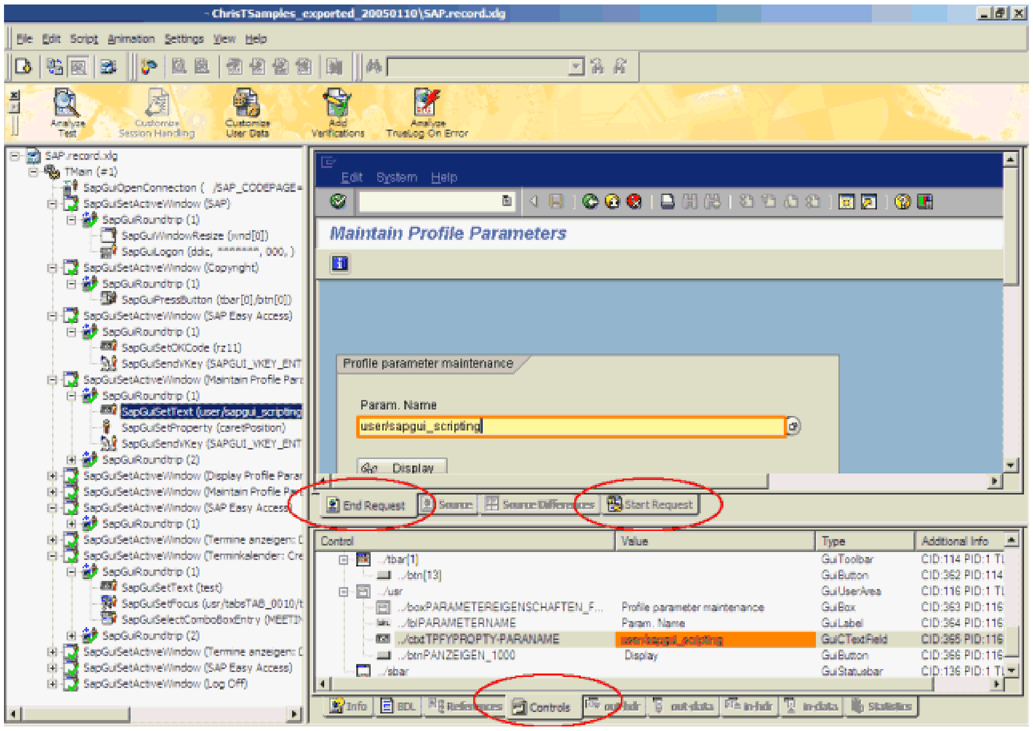
TrueLog Explorer supports the visualization of SAPGUI requests and responses in the same way it supports the visualization of HTTP client requests and HTTP/HTML server responses.

The three panes that are displayed with SAPGUI TrueLogs are:

- **TrueLog** menu tree – Lists all SAPGUI API calls included in the test run.
- **Content** pane (upper-right pane) – Displays the state of the GUI at each API node. The **End Request** and **Start Request** pages enable you to view both the initial and final states of each SAPGUI server request. These views enable you to see how the server request has affected the GUI display, such as the display of a new dialog box or error message.

  It is possible that when a window is destroyed or when a new window is created due to the action of a selected API node (for example, a button click) that the **End Request** pane will show a screenshot of the next activated window. This can be confusing as control information in the tree and the screenshot will not match. This only happens on the last node of a window.

  Note: TrueLog screenshots are captured only during Try Script runs, not full load tests.
- **Information** pane (lower-right pane) – Displays data regarding the most recent test run. The pages in this pane that are active and applicable to SAPGUI TrueLogs are **Info**, **BDL**, and **Controls**. The **Controls** page offers a convenient means of viewing and working with all customizable controls in their hierarchical order that are included on each GUI screen.



```
SAP.record.xlg
  TMain (#1)
    SapGuiOpenConnection (  /SAP_CODEPAGE=
    SapGuiSetActiveWindow (SAP)
      SapGuiRoundtrip (1)
        SapGuiWindowResize (wnd[0])
        SapGuiLogon (ddic, *******, 000, )
    SapGuiSetActiveWindow (Copyright)
      SapGuiRoundtrip (1)
        SapGuiPressButton (tbar[0]/btn[0])
    SapGuiSetActiveWindow (SAP Easy Access)
      SapGuiRoundtrip (1)
        SapGuiSetOKCode (rz11)
        SapGuiSendVKey (SAPGUI_VKEY_ENT
    SapGuiSetActiveWindow (Maintain Profile Para
      SapGuiRoundtrip (1)
        SapGuiSetText (user/sapgui_scripting
        SapGuiSetProperty (caretPosition)
        SapGuiSendVKey (SAPGUI_VKEY_ENT
      SapGuiRoundtrip (2)
    SapGuiSetActiveWindow (Display Profile Parar
    SapGuiSetActiveWindow (Maintain Profile Pa
    SapGuiSetActiveWindow (SAP Easy Access
      SapGuiRoundtrip (1)
    SapGuiSetActiveWindow (Termine anzeigen: (
    SapGuiSetActiveWindow (Terminkalender: Cre
      SapGuiRoundtrip (1)
        SapGuiSetText (test)
        SapGuiSetFocus (usr/tabsTAB_0010/t
        SapGuiSelectComboBoxEntry (MEETIN
      SapGuiRoundtrip (2)
    SapGuiSetActiveWindow (Termine anzeigen: (
    SapGuiSetActiveWindow (SAP Easy Access)
    SapGuiSetActiveWindow (Log Off)
```

Maintain Profile Parameters

Profile parameter maintenance

Param. Name: user/sapgui_scripting

Control	Value	Type	Additional Info
../tbar[1]		GuiToolbar	CID:114 PID:1 TL
../btn[13]		GuiButton	CID:362 PID:114
../usr		GuiUserArea	CID:116 PID:1 TL
../boxPARAMETEREIGENSCHAFTEN_F...	Profile parameter maintenance	GuiBox	CID:363 PID:116
../lblPARAMETERNAME	Param. Name	GuiLabel	CID:364 PID:116
../ctxtTPFYPROPTY-PARANAME	user/sapgui_scripting	GuiCTextField	CID:365 PID:116
../btnPANZEIGEN_1000	Display	GuiButton	CID:366 PID:116
../sbar		GuiStatusbar	CID:136 PID:1 TL

Under each `SapGuiSetActiveWindow` node is a virtual node that does not reflect an API call. Those nodes are called `SapGuiRoundtrip` and contain all API calls that have been sent to the server in a single roundtrip. Only the last API call under the `SapGuiRoundtrip` node forces a roundtrip. This API call also generates the measures that have been selected in the profile settings with the timer name specified as the last parameter for the call.

When you have **Capture Screenshots** enabled, a screenshot is captured with each newly activated window. Screenshots are displayed in TrueLog Explorer when you click a `SapGuiSetActiveWindow` node or a sub node.

When you also have **Capture Screenshots for every action** enabled, a screenshot is taken before and after each roundtrip. These screenshots are displayed in TrueLog Explorer when you click a `SapGuiRoundtrip` node or a sub-action node.

When the **Log control information in TrueLog** option is enabled, information about each control on each window is logged. When a control changes its value at a certain API node, the changes are reflected in the TrueLog and the controls are displayed in a tree hierarchy.

Control Information

Depending on profile settings, controls are either logged for all windows (**Log control information in TrueLog**) or only logged in the case of errors (**Log control information on error**).

The following control information is logged for each control:

- Internal unique sequence number
- Absolute and relative object/control ID
- Current value of the object
- Type of object
- Sequence number of parent control
- Screen coordinates of the control

The unique internal sequence number is for internal use only. It makes it possible to build a hierarchical view of the controls.

The current value of each object depends on the control type. For text controls, the `Text` property is used. Other controls have properties that reflect other value types.

Verification and Parsing Functions

Verifications and parsing are possible with most control types. TrueLog Explorer offers standard verification and parsing wizards for SAPGUI TrueLogs. All you need to do to access a wizard, is right-click a control in a screenshot or on the control tree. Via a context menu you can then launch the verification and parsing wizards.

Verification and parsing functions are scripted after the currently selected API node in the tree. Replay errors can result when verification and parsing functions are inserted after the last API call of a window. For example, if a currently selected API node is a button press on a **Close** button, the current window and all the controls of that window will be destroyed by this action. A verification/parsing function scripted after this call will therefore fail with a `Handler not found` error during replay. For this reason TrueLog Explorer prompts you to confirm that you wish to add verification/ parsing functions that are inserted on the last nodes of windows.

TrueLog Explorer allows you to add content checks to verify whether the content that is to be sent by servers is in fact received by clients under real-world conditions. For any SAPGUI function call in which input data is inserted, you can insert a return value verification function. Verification functions can be inserted from either **Source** view or the **Controls** menu tree.

By comparing replay test runs with record test runs, TrueLog Explorer allows you to confirm visually whether or not text, graphics, field data, and more are downloaded and displayed by clients while SAPGUI applications are under heavy load. This comparison allows you to detect a class of errors that other SAPGUI traffic simulation tools are not able to detect: Errors that occur only under load that are not detected with standard test scripts.

Content verification functions remain useful after system deployment as they can be employed in ongoing performance management.

Input Data Customization

API calls that simulate user input can be customized in TrueLog Explorer. Such API calls include `SapGuiSetText` and `SapGuiLogon`. TrueLog Explorer's **Step-through-TrueLog** dialog allows you to step through all customizable calls.

TrueLog Explorer's Parameter Wizard can be used to customize input data with:

- Constant values
- Random values from multiple sources
- Multi-column data file support
- Values from previously parsed values

Settings for Large Load Tests

Explains issues you should consider when running large load tests.

Suggested Profile Settings

Control Information

When running large load tests it is recommended that you disable the **Log control information in TrueLog** setting. This improves the overall performance of each executed virtual user. With this option enabled, all controls of each window iteration are logged to the TrueLog, which is resource (CPU) intensive.

By instead enabling **Log control information on error**, you still gain the benefit of seeing each control of the windows on which errors occur.

Screenshots

Screenshots are only captured when SAPGUI is run in visible mode. This behavior can be enabled/ disabled via the **Show SAPGUI during single runs** profile setting. When running load tests, this setting is automatically disabled regardless of profile settings. This ensures that screenshots will not be taken during load tests and thereby impact the overall performance of virtual users.

When errors occur, you can however see control information in a TrueLog On Error file if either the **Log control information on error** or **Log control information in TrueLog** option is enabled. Control information is displayed in a tree structure, which enables you to see which controls led to errors and the screen coordinates of those controls. To visually see where controls are located, open corresponding record and replay TrueLogs in compare mode and synchronize them. You can then compare the state of a control when an application was recorded with the state of that same control when the script was replayed.

Connection Handling

It is not recommended that you place `SapGuiOpenConnection` calls in main transactions, otherwise each user opens a new connection with each transaction iteration. Because the SAPGUI scripting API performs connection closes asynchronously, this leads to the problem that your load test agents will open too many connections to the SAP server simultaneously. Errors will then result if you exceed the maximum connection limit.

Unless you are interested in load testing the establishment of new connections, it is recommended that you move logon procedures to `INIT` transactions. Furthermore, main transactions should always end at the same location where they begin, which is likely to be the screen after a successful logon.

Likewise, logout sequences should be moved out of main transactions and into end transactions.

A fully customized script should resemble the following:

```
dcluser
  user
    VUser
  transactions
    TInit           : begin;
    TMain           : 1;
    TEnd            : end;
var
  gsConnID : string;

dcltrans
  transaction TInit
  begin
    // Connecting to SAP
    gsConnID := SapGuiOpenConnection("CONNECTSTRING");
    SapGuiSetActiveConnection(gsConnID);
    SapGuiSetActiveSession("ses[0]");
    SapGuiSetActiveWindow("wnd[0]");
    // Logon to SAP System
    // Before running a test you have to customize the password
    parameter!
    SapGuiIgnoreError(SAPENGINE_STATUSBAR_CHANGED, SEVERITY_SUCCESS);
    SapGuiLogon("username", "password", "000", "", "SapGuiLogon");
  end TInit;

  transaction TMain
var
  begin
    SapGuiSetActiveWindow("wnd[0]", "SAP Easy Access", SAP GUI_MATCH_
    ExactNoCase);
...
...
...
    // The VUser should now be again back on the window after a
    successful logon
  end TMain;

  transaction TEnd;
  begin
    SapGuiPressButton("tbar[0]/btn[15]", "SapGuiPressButton\\btn[15]");
    // Log Off
    SapGuiSetActiveWindow("wnd[1]", "Log Off", SAP GUI_MATCH_
    ExactNoCase);
    // Yes
    SapGuiPressButton("usr/btnSPOP-OPTION1", "SapGuiPressButton\\Yes");
  end TEnd;
```

Logon

When running load tests it is recommended that you use a different SAP user for each virtual user. This can be managed by creating users with user names that have sequence numbers appended to them, for example SAPUSER1-SAPUSER99.

In scripts, it is then possible to customize the username parameter of `SapGuiLogon` in the following way:

```
SapGuiLogon("SAPUSER" + string(GetUserId()), "PASSWORD", "000", "");
```

By default `SapGuiLogon` enters passed parameters to corresponding text fields on the logon screen and then presses the `Enter` key.

When you attempt to login a user who is already logged in, SAP opens a dialog that tells you that the user is already logged in. Three options are then offered for proceeding:

- Continue and cancel other logged in users
- Continue and do not cancel other logged in users
- Do not log in the new user

By default `SapGuiLogon` chooses the second option: logging into the system without canceling other sessions of the same user. Changing the option `SAPGUI_OPT_LOGON_BEHAVIOR` with `SapGuiSetOption` allows you to customize this behavior:

```
SapGuiSetOption(SAPGUI_OPT_LOGON_BEHAVIOR, nValue)
```

Where `nValue` can be one of 4 values:

0	Do not Handle Multi-Logon automatically This means that the dialog will not handled automatically; it will be treated as a normal new window. You can then handle the window manually.
1	No multiple Logon This means that the option Do not logon is selected.
2	Multiple Logon (default) This means that the option Continue and do not cancel other logged on users is selected.
3	Logon-Kill other sessions This means that the option Continue and cancel other logged on users is selected.

Tips and Tricks

Explains issues that need to be considered when testing SAPGUI applications with Silk Performer.

Memory Usage for SAPGUI Load Testing

The general rule of thumb for SAPGUI load tests is that you can run 50 virtual users on a 1 GB RAM and 1 GHz computer.

It is possible to execute more than 50 users on a computer if you have more power and more RAM, but the increase is not linear nor can this rule be applied to all operating systems.

The real bottleneck for SAPGUI testing is neither RAM nor CPU; it is the number of system-wide GUI resources.

The exact number of possible virtual users per computer is difficult to estimate; it depends on the scripts that are executed. If the virtual users execute transactions that contain lots of controls on the screen, it is more likely that the system runs out of GUI resources faster than with another script.

Therefore you should do some test runs to determine the exact number of users that you can simulate. Do this by executing an increasing workload. When the limit is reached you will see users that are either:

- Not able to open a new SAP connection
- Run into timeouts

Checking for Unexpected Windows

Sometimes it is possible that additional windows and dialogs that did not appear during recording appear during replay. For example, a `multiple user login` alert displays when logging in with the same user multiple times.

To create an error-free script, it is necessary to check whether a new window is available before scripting such functions as `SapGuiSetActiveWindow`, otherwise an error will occur.

The **Multiple Logon** dialog is normally handled automatically by `SapGuiLogon`. The following script shows you how to handle this situation correctly:

```
if(SapGuiVerifyWindowAvailability("wnd[1]", null, false,
SEVERITY_INFORMATIONAL)) then
SapGuiSetActiveWindow("wnd[1]");
SapGuiSelectRadioButton("");
SapGuiPressButton("");
end;
```

In addition to `SapGuiVerifyWindowAvailabiltity`, it is also possible to use `SapGuiVerifyObjectAvailability` with an absolute object ID. For example:

```
SapGuiVerifyObjectAvailability(gsConnId + "/ses[0]/
wnd[1]", true, SEVERITY_INFORMATIONAL)
```

The reason we use `SEVERITY_INFORMATIONAL` is that we do not want to receive an error if the object/window is not present, we only want to perform an action if the object is there. No error should be logged if the object is not there.

`SapGuiVerifyWindowAvailability` has an additional parameter with which the window title can be verified. This can be useful when you expect one of multiple possible dialogs where the user can proceed in one of several ways. The function can be used to simply verify if a dialog with a certain title is available.

Browsing Through Grids and Tables

Below are some examples of functions used for accessing the data in grid and table controls.

Grid Control

A grid control has columns and rows. A cell is identified by its row index and column name. To get the value of a cell you must specify the row index and the column name. The following sample shows you how to access all cells in a grid by iterating through all columns and rows.

```
SapGuiGridGetColumnCount("CTRLID", iColCount);
SapGuiGridGetRowCount("CTRLID ", iRowCount);
for rowix:=0 to iRowCount-1 do
for colix:=1 to iColCount do
SapGuiGridGetColumnName("CTRLID ", colix, colName);
SapGuiGridGetCellData("CTRLID ", rowix, colix,
cellValue);
end;
end;
```

Row indices are 0-based. Therefore, you iterate from rowix 0 to row count -1.

Table Control

The table control is similar to the grid control. A cell is identified by its row and column index. Both indices are 0-based. The following sample shows you how to access all cells in the table by iterating through all columns and rows.

```
SapGuiTableGetColumnCount("CTRLID", iColCount);
SapGuiTableGetRowCount("CTRLID ", iRowCount);
for rowix:=0 to iRowCount do
for colix:=1 to iColCount do
cellValue := SapGuiTableGetText("CTRLID ", rowix,
colix);
end;
end;
```

Accessing Control Properties

The SAPGUI scripting interface offers the ability to access all controls on the current window. Each control has different properties that can be accessed with `SapGuiGetProperty`. As SAPGUI scripting is based on COM technology, there are many COM libraries that implement those COM objects. Those libraries also include type information for each control type.

To get a list of properties and methods of the different control types, you can view those libraries with a tool that allows COM type library inspection. OLEView32 is a tool that comes with Microsoft Visual Studio that can be used for this purpose.

The following libraries can be found in the SAP install directory (usually: `C:\Program Files\SAP\Frontend`):

- `SAPGUI\sapfewse.ocx`

  This contains the basic controls (for example, text, button)
- `Controls\Scripting\*.dll`

  This contains extended controls such as trees and grids

All object properties can be accessed with `SapGuiGetProperty`. For example:

```
SapGuiGetProperty("TXTCTRLID", "text", sTextValue);
```

When a property returns another SAPGUI object or SAPGUI collection, the returned object is stored internally and can be accessed with the constant value `SAPGUI_ACTIVEOBJECT` for any `SapGui` API call for the control ID parameter. For example:

```
SapGuiPressButton(SAP GUI_ACTIVEOBJECT);
```

Invoking Control Methods

Methods on SAP controls can be invoked using `SapGuiInvokeMethod`. The basic information about methods that are offered by controls can be viewed in the COM type library. OLEView32 is a tool that comes with Microsoft Visual Studio that can be used for this purpose.

To press a button, you can either use the high-level `SapGuiPressButton` method or you can invoke the `pressButton` method on the button control, like this:

```
SapGuiInvokeMethod("BUTTONCTRLID", "pressButton");
```

When a method call returns another SAPGUI object or SAPGUI collection, the returned object is stored internally and can be accessed with the constant value `SAPGUI_ACTIVEOBJECT` for any SAPGUI API call for the control ID parameter.

Siebel Support

This section explains how to use Silk Performer to load test Siebel 8.x CRM applications. It covers the testing of Siebel thin client (Web client) installations that utilize front ends comprised of HTML and applets running within a Web browser (for example, Internet Explorer).

This chapter explains how with the Silk Performer Web Recorder:

- There is no need for session info customization.
- There is no need to manually insert parsing functions for database keys.
- Scripts work correctly, even with transactions that insert new records into databases.
- Scripts are prepared for easy randomization.

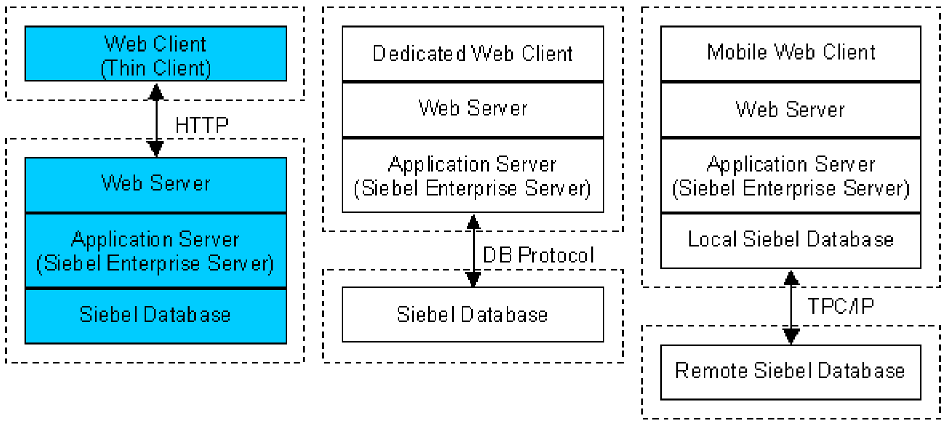
Also covered are:

- Best practices for properly preparing recording sessions
- Basic architecture of Siebel thin client installations

This section is intended for experienced users who are knowledgeable about load testing Web (HTTP/ HTML based) applications with Silk Performer.

Siebel CRM Application Architecture

This topic contains an overview of the Siebel client. The figure below illustrates the architecture of a typical Siebel deployment.



This section covers the portion of the above figure that is highlighted in blue: the Web client (thin client). This is the default deployment for Siebel CRM installations.

The Web client is an HTTP-, HTML-, Java-applet based application that does not require a client-side software installation.

The load testing of Siebel thin client installations is achieved by recording and replaying HTTP traffic generated by browsers and Java applets.

Dedicated Web clients and mobile Web clients can also be tested using Silk Performer, although those topics are not covered in this section.

HTTP Traffic

To understand recorded scripts, it is helpful to distinguish between two types of HTTP traffic:

- HTTP traffic generated by browsers
- HTTP traffic generated by Java applets

HTTP traffic generated by browsers is similar to HTTP traffic generated by typical Web applications (for example, HTML and pictures).

HTTP traffic generated by Java applets consists primarily of `POST` requests sent to servers. The bodies of HTTP requests have the MIME type `application/x-www-form-urlencoded`, which is the same MIME type used for form submissions in HTML based applications. HTTP response bodies however are in a proprietary Siebel format.

The first example below shows the body of a typical Java applet HTTP request.

HTTP Request Body from a Java Applet

```
SWERPC=1&SWECount=5&SWECmd=InvokeMethod&SWEMethod=GetPreference&
SWEInputPropSet=%400%600%603%600%60GetPreference
%603%60%60pCategory%60SWE%3AListAppletLayout%60pPrefName
%60%60SWEJSXInfo%60false%60
```

BDL Form Definition for a HTTP Request Body from a Java Applet

This example shows the corresponding BDL form definition that is generated by the Web Recorder for this HTTP request.

```
SALES_START_SWE018 <ENCODE_CUSTOM> :
  "SWERPC"               := "1",
  "SWECount"             := "5",
  "SWECmd"               := "InvokeMethod",
  "SWEMethod"            := "GetPreference",
  "SWEInputPropSet"      :=
"@0`0`3`0`GetPreference`3``pCategory` SWE:"

"ListAppletLayout`pPrefName``SWEJSXInfo`false`";
```

Setting Up Your Project

Note: The Siebel Web client does not require an explicit software installation on the client computer. The client runs within a Web browser. Your browser may however prompt you for security reasons to confirm that the Siebel Java applet should be downloaded and installed. To avoid unnecessary security warnings in the future, add the name of the Siebel Web server to your browser's list of trusted sites.

1. Select **File** > **New Project** to create your project.
2. Select **ERP/CRM** > **Siebel** > **Siebel Web Client**
3. Click **Next**. The Recorder is displayed, ready to begin recording your interactions with the Siebel application.
4. *(optional)* If the Siebel installation under test uses load balancing through varying server names or requires additional context management, go to **Settings** > **Active Profile** > **Web** > **Recording** and adjust Web context management settings.

Scripts that use the host name `standardhost` in place of a real host name can be replayed against different servers by changing the host name at **Settings** > **Active Profile** > **Internet** > **Hosts**. The recorder can generate such scripts only if the host name is changed before recording takes place.

Configuring TrueLog Explorer

To ensure best results on the **Differences** pane within TrueLog Explorer you must add some characters to both the **Compare tags** and **Compare tags HTML** lists.

1. Within TrueLog Explorer, go to **Settings** > **Options** > **Compare Tags**.

2. Enter the following values into the table rows:
   1. & (ampersand)
   2. * (asterisk)
   3. ` (reverse single quote)
   4. @ (at)
3. Go to **Settings** > **Options** > **Compare Tags HTML** and enter the same table row values there.

Dynamic Information in Web Client HTTP Traffic

This section explains that HTTP traffic generated by Siebel Web clients incorporates dynamic information that must be addressed to ensure successful and accurate replay.

Error Detection

Failure to handle dynamic data does not always generate error messages during script replay. This is because the Siebel Web Server does not use HTTP error status codes to indicate errors. Instead, it sends error notifications to the applet, which in turn displays an error message indicating the condition.

This may lead to a false impression of successful replay when in fact load generated on the database tier of the application was different, more than likely less, than it should have been.

The example below shows an HTTP response body sent from a server to a Java applet that describes an error condition. Such a response is returned with the HTTP status code `200 OK`, falsely suggesting that the HTTP request was handled successfully.

Error Condition Response Sent from Server to Applet

```
@0`0`3`2``0`UC`1`SWECount`10`Status`Error`0`1`Errors`0`2`0`Level
0`0`ErrMsg`An error happened during restoring the context for
requested location`ErrCode`27631`0`0`Notifications`0`
```

Application Level Error Detection in Recorded Scripts

Silk Performer scripts can detect such application level errors. The Web Recorder generates scripts that contain such checks by adding the lines highlighted in the example below:

```
use "WebAPI.bdh"
use "SiebelWeb.bdh"

// ...

  transaction TInit
  begin
    WebSetBrowser(WEB_BROWSER_MSIE55);
    WebFormDefineEncodeCustom("base=ENCODE_FORM;"
                              " +'@'; -'!()$,'; -'''");
    SiebelWebInit();
  end TInit;
```

Session IDs

Siebel Web Server uses session IDs to track user sessions. Servers can be configured to send session IDs either in cookies or in URLs (form fields or query strings; this is the default configuration).

The example below shows an HTTP response header with a `Set-Cookie` header for a session ID. Session IDs in cookies are handled automatically by Silk Performer.

Siebel Session ID in a Cookie

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 12 Mar 2002 17:26:01 GMT
Content-Language: en
Cache-Control: no-cache
Content-Type: text/html;charset=windows-1252
Content-Length: 3043
Set-Cookie: _sn=!1.428.556d.3c8e3a29; Version=1; Path=/sales
```

Siebel Session ID in a URL

This example shows a fragment of an HTML frameset with a frame that incorporates a session ID in a URL.

```
<frameset>
  <frame name="_sweclient">
  <frame name="_swe" src="http://lab61/sales/start.swe?
                        SWEFrame=top._swe&SWECount=1&
                        _sn=!1.6c0.447b.3ceccd1b
                        &SWECmd=GetCachedFrame">
</frameset>
```

The Silk Performer Web Recorder generates scripts that accurately handle such session IDs.

Fragments of a Recorded Script with Session IDs

This example shows fragments of a recorded script that handles the above session ID using the following techniques:

- A variable `gsSid` is declared to hold the session ID.
- A parsing function `WebParseDataBoundEx` is inserted before the function call `WebPageLink` for parsing of the session ID, which is included in the server response of the `WebPageLink` function (see example above).
- The parsed result can be used wherever it is required. In this example, it is required for a form submission.
- Informational comments enhance readability.

```
var
  gsSid : string; // !1.6c0.447b.3ceccd1b

// ...

WebParseDataBoundEx(gsSid, sizeof(gsSid), "&_sn=", 1,
                    "&", WEB_FLAG_IGNORE_WHITE_SPACE);
WebPageLink("replace", "Siebel Sales (#2)", 1, "_sweapp");

WebPageForm("http://lab61/sales/start.swe",
            SALES_START_SWE003, "Unnamed page (#2)");

// ...

dclform
  SALES_START_SWE003 <ENCODE_CUSTOM> :
    "SWERPC"     := "1",
    "SWECount"   := "1",
    "_sn"        := gsSid,// value: "!1.6c0.447b.3ceccd1b"
    "SWEJSXInfo" := "false",
    "SWECmd"     := "InvokeMethod",
    "SWEService" := "SWE Command Manager",
    "SWEMethod"  := "PrepareGlobalMenu";
```

Time Stamps

HTTP communication between Java applets and the Siebel Web Server includes timestamps, which are strings that tell the number of milliseconds since Jan 1, 1970. Proper replays must include correct timestamps. The Silk Performer kernel API (file `kernel.bdh`) provides the function `GetTimeStamp()`, which is used to obtain accurate timestamp strings.

The Web Recorder recognizes timestamps and generates scripts that use them.

HTTP Request Body with Timestamp

This example shows the body of a Java applet HTTP request that incorporates a timestamp.

```
SWEUserName=undisclosed&SWEPassword=undisclosed&SWEForm=SWEEntry
Form&SWENeedContext=false&SWECount=0&SWECmd=ExecuteLogin&SWETime
Stamp=1023377037797
```

Corresponding BDL Form with Timestamp

This example shows the corresponding BDL form generated by the Web Recorder. The value of the timestamp has been replaced by a call to the function `GetTimeStamp()`.

```
dclform
  SWEENTRYFORM002:
    "SWEUserName"    := "undisclosed", // changed
    "SWEPassword"    := "undisclosed", // changed
    "SWEForm"        := "SWEEntryForm", // added
    "SWENeedContext" := "false", // added
    "SWECount"       := "0", // added
    "SWECmd"         := "ExecuteLogin", // added
    "SWETimeStamp"   := GetTimeStamp(); // added, value:
"1023377037797"
```

URL Encoding

According to HTTP specifications, unsafe characters included in transmitted data with the MIME type `application/x-www-form-urlencoded` must be encoded (escaped). This is achieved by replacing unsafe characters with hexadecimal equivalents and preceding them with percent (%) symbols.

Silk Performer provides four standard encoding types: `ENCODE_FORM, ENCODE_ESCAPE, ENCODE_BLANKS and ENCODE_NONE`. These encoding types differ in terms of the characters they escape.

Siebel Web client Java applets apply a unique encoding type that differs from the standard Silk Performer encoding types. For this reason, Silk Performer provides a new encoding type, `ENCODE_CUSTOM`, which can be configured using the function `WebFormDefineEncodeCustom`.

Siebel uses an encoding type that differs from the standard encoding type `ENCODE_FORM` in the following respects:

- Characters not escaped: `!()$,'`
- Characters escaped: `@`

The Silk Performer Web Recorder detects when an HTTP request applies this special encoding type, and then generates a script using the following techniques:

- Function `WebFormDefineEncodeCustom()` is used to define the encoding type `ENCODE_CUSTOM` in the `TInit` transaction.
- The encoding type `ENCODE_CUSTOM` is used in form definitions.

Fragments of a Recorded Script that utilizes ENCODE_CUSTOM

This example shows fragments of a recorded script that utilizes these techniques.

```
dcltrans
  transaction TInit
  begin
    WebSetBrowser(WEB_BROWSER_MSIE55);
    WebFormDefineEncodeCustom("base=ENCODE_FORM;"
                              " +'@'; -'!()$,'");
  end TInit;

// ...

dclform
  SALES_START_SWE011 <ENCODE_CUSTOM> :
    "SWERPC"                  := "1",
    "SWECount"                := "4",
    "SWECmd"                  := "InvokeMethod",
    "SWEMethod"               := "GetPreference",
    "SWEInputPropSet"         :=
"@0`0`3`0`GetPreference`3``pCategory`"

"Behavior`pPrefName``SWEJSXInfo`false`";
```

User Input

Generally, it is desirable if user input recorded during recording sessions can be easily identified and changed (for example, randomized) in recorded scripts.

This is especially true for Siebel Web client scripts because Java applets often send values for all input fields to the server. If a new database record is created (for example, for a new customer) and the record is subsequently edited or viewed, input (for example, customer name) may appear multiple times in recorded scripts.

The Silk Performer Web Recorder:

- Detects user input by assuming that form field values are user input when one of the following is true:
  - Values are enclosed in $ symbols
  - Values are enclosed in underscores`(_)`
  - Values begin with the character sequence `i.`
- Generates a variable for each user input and uses that variable in a script, in place of the original value.

Fragments of a Recorded Script that Includes User Input

```
var
  gsInputNewName     : string init "$NewName$";
  gsInputNewSite     : string init "_NewSite_";
  gsInputiHttp_x_com : string init "i.http://x.com";

// ...

dclform
  SALES_START_SWE020 <ENCODE_CUSTOM> :
    "SWEMethod"   := "GetQuickPickInfo",
    "SWEViewId"   := "",
    "SWEView"     := "Account List View",
    "SWEApplet"   := "Account List Applet",
    "SWEField"    := "s_1_2_46_0",
    "SWERow"      := "0",
    "SWEReqRowId" := "1",
    "s_1_2_38_0"  := "N",
    "s_1_2_39_0"   := gsInputNewName, // value: "$NewName$"
    "s_1_2_40_0"   := gsInputNewSite, // value: "_NewSite_"
    "s_1_2_41_0"  := "(999) 999-9123",
    "s_1_2_37_0"  := gsInputiHttp_x_com,
                        // value: "i.http://x.com"
    "s_1_2_49_0"    := "",
    "s_1_2_46_0"    := "",
    "s_1_2_44_0"    := "",
    "SWEBCVals"     :=
"@0`0`0`1``3``2`0`FieldValues`3``FieldArray"
                    "`4*Name8*Location17*Main Phone Number"
                    "`ValueArray`14*" + gsInputNewName +
                    "14*" + gsInputNewSite +
"10*9999999123`";
```

Choose input values during recording based on these descriptions so that the Web Recorder will detect them.

Dynamic Data

Siebel HTTP traffic includes a variety of dynamic data that are sent to servers within HTTP requests. Such data can be parsed from previous server responses and substituted into scripts in place of hard-coded values.

This includes:

- Existing database record values
- Row IDs

Siebel uses database keys to identify records in databases. Such keys are present in HTTP traffic emanating from both browsers and applets. Database keys are also known as Row IDs. Accurate handling of row IDs and other dynamic data is essential for successful replay.

Row IDs and other dynamic data can be included in HTML documents (see first example below) or in responses to HTTP requests from Java applets (see second example below).

Dynamic data are always contained in value arrays (lists of values for single rows to be displayed in the Siebel GUI).

Value Array in JavaScript Code, within a HTML Document

```
<script>
  row = new top._swescript.JSSRecord();
  row.id = "1-9Q1";
  row.valueArray = ["Foo","Bar","1234567890","","Active",
                    "http://www.foo.com/bar","","","USD",
                    "11/26/2002","","USD","11/26/2002",
                    "","","","","N","N","","1-9Q1"];
  S_BC1.Rec(row);
</script>
```

Value Array in an Applet Response

```
@0`0`3`2``0`UC`1`Status`Completed`SWEC`10`0`24`Notifications`0`2
`0``0`OP`bn`bc`S_BC1`7`0``0`type`SWEIRowSelection`OP`g`br`0`cr`6
`bc`S_BC1`size`7`ArgsArray`20*Account Entry
Applet1*1`7`0``0`type`SWEIRowSelection`OP`g`br`0`cr`6`bc`S_BC1`s
```

```
ize`7`ArgsArray`19*Account List
Applet1*11*01*01*01*01*01*0`7`0``0`type`SWEIPrivFlds`OP`g`br`0`c
r`6`bc`S_BC1`size`7`ArgsArray`19*Account List
Applet11**BlankLine11*?11**BlankLine21*?9**HTML URL1*?15**HTML
RecNavNxt1*?
`7`0``0`type`SWEICanInvokeMethod`OP`g`br`0`cr`6`bc`S_BC1`size`7`
ArgsArray`19*Account List
Applet1*01*11*11*01*21*11*31*01*41*11*51*11*61*11*71*11*81*11*91
*12*101*02*111*02*121*12*131*02*141*12*151*12*161*12*171*12*181*
12*191*12*201*12*211*12*221*12*231*12*241*12*251*02*261*12*271*1
2*281*02*291*02*301*02*311*02*321*02*331*02*341*12*351*12*361*1`
7`1``0`OP`iw`index`7`br`0`cr`6`bc`S_BC1`size`7`ar`0`1`0`FieldVal
ues`0`ValueArray`3*Foo3*Bar10*12345678900*6*Active22*http://
www.foo.com/
bar0*0*3*USD10*11/26/20020*3*USD10*11/26/20020*0*0*0*1*N1*N0*5*1
-9N9`8`0``0`OP`dw`index`7`br`0`cr`6`bc`S_BC1`nr`1`size`7`ar`0`8`
0``0`value`0`OP`sc`br`0`cr`6`bc`S_BC1`size`7`ar`0`state`n`8`0``0
`value`1`OP`sc`br`0`cr`6`bc`S_BC1`size`7`ar`0`state`n`8`0``0`val
ue`7`OP`sc`br`0`cr`6`bc`S_BC1`size`7`ar`0`state`cr`8`0``0`value`
1`OP`sc`br`0`cr`6`bc`S_BC1`size`7`ar`0`state`nrk`8`0``0`value`13
`OP`sc`br`0`cr`6`bc`S_BC1`size`7`ar`0`state`nr`6`0``0`OP`nd`br`0
`cr`6`bc`S_BC1`size`7`ar`0`2`0``0`OP`en`bc`S_BC1`0`3`
```

Dynamic information typically appears in the `dclform` section of scripts.

When required, the Web Recorder automatically generates a parsing function (`WebParseDataBoundEx`) and substitutes parsed values wherever they appear in scripts. Parsing functions parse for complete value arrays in HTML and applet responses. The Recorder uses an appropriate tokenizing function `SiebelTokenHtml` or `SiebelTokenApplet` to retrieve individual tokens from parsed value arrays.

Script Fragment Utilizing Parsing Functions

This example shows fragments of a recorded script that utilizes the following techniques:

- Variables are declared for the value array (gsRowValArray and gsRowValArray_001).
- Parsing functions WebParseDataBoundEx are inserted to parse the value arrays, which are included in the server response of the subsequent function (see examples above).

The parsed result can be used wherever it is required. In this example, individual tokens of the parsed values occur in various locations in form definitions. Informational comments are used to enhance readability.

```
var
  gsRowValArray     : string; // =
["Foo","Bar","1234567890","","Active",
                              //    "http://www.foo.com/
bar","","","USD",
                              //
"11/26/2002","","USD","11/26/2002",
                              //
"","","","","N","N","","1-9Q1"];
  gsRowValArray_001 : string; //
3*Foo3*Bar10*12345678900*6*Active
                              // 22*http://www.foo.com/bar
                              //
0*0*3*USD10*11/26/20020*3*USD10*11/26/2002
                    // 0*0*0*0*1*N1*N0*5*1-9N9

// ...
```

```
  WebParseDataBoundEx(gsRowValArray, sizeof(gsRowValArray),
            "row.valueArray", 2, "S_",
WEB_FLAG_IGNORE_WHITE_SPACE, 1);
// function call where parsing function is in effect

  WebParseDataBoundEx(gsRowValArray_001,
sizeof(gsRowValArray_001),
                     "ValueArray`", 1, "`",
WEB_FLAG_IGNORE_WHITE_SPACE, 1);
// function call where parsing function is in effect

// ...

dclform

// ...

SALES_ENU_START_SWE016 <ENCODE_CUSTOM> :
  "SWEMethod"  := "Drilldown",
  "SWEView"    := "Account List View",
  "SWEApplet"  := "Account List Applet",
  "SWEReqRowId":= "1",
   "s_1_2_40_0"      := SiebelTokenHtml(gsRowValArray, 0), //
value: "Foo"
   "s_1_2_41_0"      := SiebelTokenHtml(gsRowValArray, 1), //
value: "Bar"
  "s_1_2_42_0" := "(123) 456-7890",
  "s_1_2_51_0" := "",
   "s_1_2_47_0"      := SiebelTokenHtml(gsRowValArray, 4), //
value: "Active"
  "s_1_2_45_0" := SiebelTokenHtml(gsRowValArray, 5),
                       //value:"http://www.foo.com/bar"
  "SWECmd"     := "InvokeMethod",
  "SWERowId"   := SiebelTokenHtml(gsRowValArray, 20), // value:
"1-9Q1"
  "SWETS"      := GetTimeStamp(); // value: "1038305654969"

// ...

SALES_ENU_START_SWE022 <ENCODE_CUSTOM> :
  "SWEMethod"  := "Drilldown",
  "SWEView"    := "Account List View",
  "SWEApplet"  := "Account List Applet",
  "SWEReqRowId":= "1",
   "s_1_2_40_0"   := SiebelTokenApplet(gsRowValArray_001,
0), // value: "Foo"
   "s_1_2_41_0"   := SiebelTokenApplet(gsRowValArray_001,
1), // value: "Bar"
  "s_1_2_42_0" := "(123) 456-7890",
  "s_1_2_51_0" := "",
  "s_1_2_47_0" := SiebelTokenApplet(gsRowValArray_001, 4), //
value: "Active"
  "s_1_2_45_0" := SiebelTokenApplet(gsRowValArray_001,5),
                                    // value: "http://
www.foo.com/bar"
  "SWECmd"     := "InvokeMethod",
  "SWERowId"   := SiebelTokenApplet(gsRowValArray_001, 20), //
value: "1-9N9"
  "SWETS"      := GetTimeStamp(); // value: "1038305711331"
```

Reformatting Functions

The Siebel Web client reformats phone numbers and date/time values that are sent by the server or entered by users. Therefore the format of such values in client-sent HTTP request bodies is different from the format that is used in server-sent HTTP response bodies.

Phone Number in Server-Response Body

```
10*987140255510*78140110000*0*0*22*main.contact@MyCompany.com5*1
-FIH
```

Phone Number in Recorded Script, Reformatted by the Siebel Web Client

```
"s_1_1_25_0" := "(987) 140-2555";
```

Without modification, the format used in request bodies would also be used in recorded scripts, making it impossible to replace such values with parsed values.

Reformatting Function for Phone Numbers

To allow such reformatted strings to be replaced with parsed values, the Silk Performer Web recorder records an appropriate reformatting function that mimics the Siebel Web client's reformatting and records actual values in the same format that is used in server responses.

```
"s_1_1_25_0" := SiebelPhone("9871402555");
```

Using a Parsed Value as a Parameter of a Reformatting Function

Unless a value is an input value, values can generally be replaced with parsed values in a second step. The Recorder actually records a parsed value instead of a hard-coded string.

```
"s_1_1_25_0" := SiebelPhone(SiebelTokenApplet(gsParsed, 5));
```

The following reformatting functions are available and are automatically recorded as required:

- SiebelPhone
- SiebelDate
- SiebelTime
- SiebelDateTime
- SiebelDecodeJsString
- SiebelParam

Meaningful Timer Names

The Recorder extracts meaningful timer names from form fields. This makes it easier for human readers to interpret scripts, TrueLogs, and performance reports.

Meaningful Timer Names

Here is an example of an intuitive timer name.

```
WebPageForm("http://standardhost/sales_enu/start.swe",
            SALES_ENU_START_SWE022,
            "Account List Applet: InvokeMethod: Drilldown");

// ...

dclform
```

```
// ...

SALES_ENU_START_SWE022 <ENCODE_CUSTOM> :
  "SWEMethod"  := "Drilldown",
  "SWEView"    := "Account List View",
  "SWEApplet"  := "Account List Applet",
// more form fields
  "SWECmd"     := "InvokeMethod",
  "SWERowId"   := SiebelTokenApplet(gsRowValArray_001, 20),
  "SWERowIds"  := "",
  "SWEP"       := "",
  "SWETS"      := GetTimeStamp(); // value: "1038305711331"
```

Best Practices

This topic explains the common pitfalls, offers hints, and explains best practices for successfully testing Siebel Web clients.

Read-Only Transactions

A read-only transaction is a transaction in which new records aren't inserted and data is not altered. Recorded scripts for read-only transactions run without modification.

Such scripts may however contain parsing functions for value arrays. Parsing functions may, for example, parse the value array of the third record in a table. If during replay this table contains fewer than 3 records, no value array can be parsed and the result may be replay errors. Such situations can be avoided by adjusting the occurrence parameter of parsing functions so that existing value arrays are parsed.

The better solution however is to ensure that each table in a database contains adequate records during load tests, thereby avoiding the problem altogether.

Read/Write/Update Transactions

A writing transaction is a transaction that inserts new records or changes the values of existing records.

Required Script Customizations

Scripts for writing transactions require only minimal customization to ensure successful replay. This is because value arrays for new records can be parsed from server responses.

When modifications are required for successful replay, it is generally because many tables in Siebel do not allow duplicate records. The circumstances by which Siebel considers records to be duplicates vary between tables. In most instances, a certain combination of record fields must be different for Siebel to consider a record unique.

For successful replay, these fields must be changed, even with Try Script runs. This can be done easily when the Web Recorder creates these values as variables.

For Try Script runs, it is sufficient to manually change these values directly in scripts. For tests however, data files must be prepared and scripts must be customized in such a way that each virtual user inserts records with unique value combinations.

Be Aware of Empty Tables

HTTP traffic generated by the insertion of records into empty tables differs significantly from HTTP traffic generated by inserting records into tables that are already populated with one or more records. For this reason, scripts that record the insertion of records into empty tables can not later be used during replay sessions once tables have been populated - and vice versa.

To avoid such problems, ensure that you do not insert records into empty tables during recording sessions. Also ensure that tables are not emptied before or during load tests.

Note that the following scenarios are not affected by these constraints:

- Creating new records in tables that include other records (for example, accounts).
- Creating new records in a table associated with the records just created (for example, creating new notes for new accounts). While the notes table for newly created accounts is empty, the notes table of newly created accounts during script replay is also empty, so such scripts will not cause problems.

Microsoft Silverlight Support

Prerequisites

Transformation is enabled for HTTP requests and responses that have the HTTP header `Content-Type` set to `"application/soap+msbin1"` or `"application/msbin1"`. If you need to transform data with a different HTTP Content-Type header, see *Transformation of Custom Content-Types*.

Setting up a Microsoft Silverlight Project

The workflow for testing Microsoft Silverlight browser-based applications with Silk Performer is the same as the workflow used for testing Web-based applications, with the following exception:

- On the **Outline Project - Workflow** dialog, you must select `Silverlight` under the **Web browser** node.

Terminal Emulation Support

Silk Performer's support for terminal emulation applications enables the recording of terminal-emulator traffic based on the Telnet protocol ("green screen" applications). Supported terminal types include VT100 and VT200 (UNIX, IBM AS400) and IBM mainframes accessed via TN3270(E) & TN5250.

A prerequisite for recording terminal-emulator traffic is an installed terminal-emulator for accessing host applications, while the recorder records the telnet traffic which results. Many such emulators are available on the market (for example, Rumba).

Note: For additional details regarding Silk Performer support for terminal emulation applications, please see `Miscellaneous Tutorials`, a PDF that accompanies your Silk Performer installation at **Documentation** > **Tutorials**.

Script Customization

For terminal-emulation script customization details, please see TrueLog Explorer Help.

Testing IBM Mainframe Protocols

The **Host Screen Info** page of terminal emulation TrueLogs offers controls for all fields and statuses that appear in recorded and replayed screens. These controls offer easy access to field-value and value-status parsing/verification functions.

IBM terminals rely on EBCDIC code pages that vary based on locality. Silk Performer chooses an EBCDIC code page among the code pages installed on the system. If national language characters are not displayed correctly it is possible to change the code page during project configuration.

IBM code pages control translation of human-readable ASCII input data (for example, search parameters) into and out of binary hexadecimal representation. This translation process facilitates script customization.

Creating a Terminal Emulation 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, expand the **Terminal Emulation** application type node and select the terminal emulation type you need.
5. Click **Next** to create a project based on your settings.

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

Configuring Terminal Emulation Profile Settings

1. In Silk Performer, expand the **Profiles** node in the **Project** tree.
2. Right-click the profile that you want to configure and choose **Edit Profile**.

   Tip: Alternatively, you can choose **Settings** > **Active Profile** from the menu bar.

   The **Profile - [<profile name>]** dialog box opens, and the **Replay** category is displayed in the shortcut list on the left.
3. In the shortcut list on the left, click the **Record** button. The Record category is displayed.
4. In the shortcut list, click the **Terminal Client** icon.
5. Click the **Telnet** tab.
6. In the **Telnet settings** area, specify one of the following Telnet detection modes:
   - Click the **Always assume Telnet-mode** option button to instruct the recorder to assume that each connection is a Telnet connection. Use this option for troubleshooting purposes only.
   - Click the **Auto-detect Telnet-mode** option button to instruct the recorder to automatically detect Telnet on each new connection (default).
   - Click the **Never assume Telnet-mode** option button to instruct the recorder to assume that no connections are Telnet connections. Use this option for troubleshooting purposes only.
7. In the **Command prompt string** text box, specify a command-prompt string that the recorder uses to script `WebTcpipRecvPacketsUntilData` functions wherever possible.

   This string increases the effectiveness of recorded scripts for dynamic content.
8. In the **Terminal properties** section, select one of the following items from the **Log level** list box:
   - `None`
   - `Error`
   - `Normal`
   - `Debug`
9. Ensure that the specific terminal type you are testing is displayed in the **Terminal type** list box. This setting was configured automatically when you selected the application type for the project.

   When correctly specified, the recorder scripts the appropriate terminal-type initialization functions and renders the screens to the TrueLog.
10. In the **Configuration string** text box, initialize the terminal emulator with a custom configuration string.

    For available options, refer to your terminal emulator documentation.
11. Specify the following **Screen** values to indicate the appearance of the terminal screen to the recorder:
    - Type a value in the **rows** text box to indicate the height of the terminal screen. An empty value lets Silk Performer detect the height during session initialization.
    - Type a value in the **columns** text box to indicate the width of the terminal screen. An empty value lets Silk Performer detect the width during session initialization.

- Check the **Colors** check box to specify the default color to use on the terminal screen.

   The supplied values overrule the default values that Silk Performer may detect. If your terminal application uses a custom screen size, make sure to set these values before you record a script in order to replay the script without errors.

12. In the **Host code page** text box, specify the code page that the server uses.

    The code page is used for various data conversions, and all available code pages have been installed with the system. If the required code page is not listed, you can install it. Ensure that the selected code page is also available on each agent computer. To review all installed code pages, or to enable a specific code page, go to **Start** > **Control Panel** > **Regional and Language Options** > **Advanced** . Only the host code pages that are listed and enabled with a checkmark are available.
13. Click the **Advanced** tab and check the **Record basic functions** check box to prevent the recorder from scripting synchronized receive-functions, such as `WebTcpipScreenRecvPacketsUntilCursor` and `WebTelnetScreenRecvRecordsUntilStatus`.

    If this option is enabled, scripted receive functions do not rely on rendered screen content. This option is enabled by default when a terminal type is not specified.
14. Click **OK** to save your settings.

Setting Up a Terminal Emulation Recording Profile

1. From the **Settings** menu, select **System**.
2. Click the **Recorder** button.
3. Click the **Add** button.
4. On the **Recording Profile** dialog, give the profile a name in the **Profile name** field.
5. Click **Browse** and select the .exe file of the terminal emulator on your local system.

   Note: All other settings on this dialog are standard for all TCP/IP-based applications.
6. If you are using RUMBA as your terminal emulator, check the **Record executable is different from application path** check box.
7. Enter `WDDspPag.exe` in the **Record executable** field.
8. Check the **TCP/IP** check box (note that Silk Performer TCP/IP-level API facilitates terminal-emulation application support).
9. Click **Web Settings**. The **Web Settings** dialog box opens.
10. From the **WinSock** list box, select `ws2_32.dll`.
11. Click **OK** to close the **Web Settings** dialog box.
12. Click **OK** to close the **Recording Profile** dialog box.

The new profile now appears on the **Recording Profiles** page.

Recording Terminal Emulation Applications

1. Click **Model Script** on the workflow bar. The **Workflow - Model Script** dialog box appears.
2. From the **Recording Profile** menu tree, select your terminal-emulation recording profile.
3. Enter the location of your terminal-emulation application in the **Command line** field.
4. Click **Start recording**. The Silk Performer Recorder dialog opens in minimized form, and the client application starts.
5. Using the client application, conduct the kind of interaction with the target server that you want to simulate in your test. The interaction is captured and recorded by the Recorder. A report of your actions and of the data downloaded appears on the **Actions** page.
6. When recording is complete, close the application, close the recorder, and save the test script.

7. To confirm that your test script runs as expected, execute a Try Script run in animated mode; you will then be able to view the results of the test run in TrueLog Explorer.

Terminal Emulation Session Customization

Session customization of terminal emulation application test scripts is normally not required. If session customizations are required, they must be partly performed manually. However, TrueLog Explorer can be used to insert parsing functions into test scripts.

In some cases the synchronization between replay and a host application may not work well, for example if the requested screen is not complete but the cursor position already assumes a completed screen. A recorded cursor synchronization function in the script will be satisfied with the first portion of the screen and continue executing the script out of synchronization, which may lead to replay errors. In such a case you would have to replace the recorded synchronization function(s) manually with a more suitable synchronization function in the script (for example, `WebTcpipRecvPacketsUntilData` or `WebTelnetRecvPacketsUntilData`).

TCP/IP Based Protocol Support

This section explains how to develop load test scripts for applications that use legacy or custom TCP/IP based protocols. It explores the challenges that are commonly encountered in this type of testing: dealing with dynamic server responses, translating between character code tables, string manipulation of binary data, and more.

Three protocols are examined in detail: Telnet, TN3270e, and a custom TCP/IP based protocol. Using recorded scripts from actual load test projects as examples, you will be guided through the process of recording an application, analyzing traffic, configuring recording rules, and customizing a script.

Overview

Note: The following protocols are directly supported by Silk Performer and do not require the types of customizations that are described in this section. Protocols not listed below, including so called legacy systems, do require script customization:

- TN3270
- TN5250
- VT100
- VT200+

All of the protocols discussed in this section are based on TCP/IP. With Silk Performer, you can record protocols at the TCP level and customize the resulting scripts using `WebTcpip` functions. With Silk Performer enhanced TCP/IP support, TrueLog Explorer, and rule-based recording, the process is straightforward.

This section has two objectives:

- To take you through the process of analyzing recorded custom TCP/IP traffic and give you the generic tools and methodology you need to customize such traffic.
- To analyze two protocols in-depth: Telnet and TN3270e.

Memory Usage of TCP/IP Virtual Users

Each virtual user running in its own process requires approximately 1.6 MB system memory.

Running more than one user per process results in the following memory consumption:

- **25 vusers:** Approx. 0.2 MB per vuser
- **50 vusers:** Approx. 0.1 MB per vuser

Script Customization

This section covers the challenges that are typically faced when customizing TCP-based load test scripts: dynamically receiving and verifying server responses, character conversion, and string manipulation of binary data.

Character Conversion

Following is a sampling of TCP/IP traffic recorded by Silk Performer:

```
WebTcpipSendBin(hWeb0,
    "\h00000000007DD7D311D57CA2899392F0"        // ·····}×Ó·Õ|¢···ð
    "\hF0F14011D74CA2899392F0F0F140FFEF", 32);  // ðñ@·×L¢···ððñ@·ï
WebTcpipRecvExact(hWeb0, NULL, 645);
```

Customization of such a script may at first appear daunting. The server responses (not shown in the example) look much like the client request. They do not include a single readable character.

The reality is that customization of such a script is fairly simple. The server is an IBM mainframe that uses the EBCDIC (Extended Binary Coded Decimal Interchange Code) character set rather than the more familiar ASCII character set. There is a simple one-to-one mapping between these two character sets, which can be implemented as a Silk Performer function.

Other possible causes for traffic that contains unreadable strings:

Reason	Consequence
Traffic uses a character set other than ASCII.	Character mapping functions can easily be implemented in Silk Performer BDL language.
Traffic is encrypted with a custom encryption algorithm (for example, one that does not rely on SSL, which is automatically supported by Silk Performer).	The encryption/decryption routines should be re-implemented in BDL, or made accessible to Silk Performer using a DLL or a Silk Performer framework (for example, Java Framework).
Traffic does not contain any strings, only numbers sent as binaries.	In such instances you do not have to worry about strings.

Dynamically Receiving Server Responses

Full control of client requests can be easily achieved however anticipating appropriate server responses can be challenging. Most significantly, virtual users must know when server responses are complete so that they can proceed with subsequent requests, or raise errors if server responses contain errors or aren't complete.

The Silk Performer recorder doesn't know in advance about the semantics of the TCP based protocols it records. Therefore, all server responses are recorded as follows:

```
WebTcpipRecvExact(hWeb0, NULL, 26);
```

During replay, the virtual user expects to receive exactly 26 bytes from the server. It then continues on with the next statement in the script. If the server sends more then 26 bytes, the bytes are received by the next `WebTcpipRecvExact` statement (making all further script execution unpredictable). If the server sends fewer bytes, `WebTcpipRecvExact` will report a timeout error.

Therefore this line can remain unchanged only if the number of response bytes won't change under any circumstances. If response length cannot be guaranteed, scripts must be customized so that they can handle server responses of varying length.

Appropriate customization depends on the semantics of the protocol. Three common scenarios are reflected by Silk Performer API functions, which simplify the customization process.

Case I: Packet Length Contained in the Packet

The length of request and response data may be encoded into packets at a defined position (typically in the packet header). The Silk Performer function `WebTcpipRecvProto` and its sibling, `WebTcpipRecvProtoEx`, can adequately handle such situations. They allow for definition of the position and length of packet-length information in response data. For example, the following BDL code line will be used if the length of the response data remains a two-byte sequence in the first two bytes (for example, position 0) of the server response:

```
WebTcpipRecvProto(hWeb, 0, 2, TCP_FLAG_INCLUDE_PROTO,
sData, sizeof(sData), nReceived);
```

Case II: Termination Byte Sequence

In this scenario data packets are terminated by a constant byte sequence. The corresponding Silk Performer function is `WebTcpipRecvUntil`. If, for example, the end sequence is defined by the two-byte sequence `0xFF00`, the following line would be appropriate:

```
WebTcpipRecvUntil(hWeb, sResp, sizeof(sResp), nRecv,
 "\hff00", 2);
```

Case III: No Information on Response Packet Size

This is the most challenging of the three scenarios. You can use a combination of the functions `WebTcpipRecv`, which receives a buffer of unknown length from the server, and `WebTcpipSelect`, which checks whether or not a subsequent `WebTcpipRecv` operation on the provided connection handle will block or succeed immediately.

Rule Based Recording

An advanced feature offered by Silk Performer is rule-based recording. You can configure the TCP/IP recorder to be aware of the semantics of proprietary TCP/IP protocols. In particular, the recorder can be configured for two of the scenarios discussed earlier (Packet length contained in the packet and termination byte sequence).

As a result, the recorder can automatically generate correct `WebTcpipRecvProto(Ex)` and `WebTcpipRecvUntil` functions so that further customization for this part of the script isn't required.

When configuring recording rules, you write a configuration file encoding the recording rules in XML and save them to the project's `Documents` folder (project specific rules) or in the public or the user’s `RecordingRules` directory (global rules). Recording rule files carry the file extension `.xrl`.

Rule-based recording exists for TCP/IP and HTTP, however only recording rules for TCP/IP are discussed in this section.

String Manipulation of Binary Data

Request Parameterization

Example code contained in a recording of a TCP session:

```
WebTcpipSendBin(hWeb0, "\h0000104F0000005065746572FFFF",
14); // ···O···Peter
```

The actual business data sent here, the data that should be parameterized, is `"Peter" = 0x50 65 74 65 72`, but that will not work here. `"\h0000104F000000" + sName + "\hFFFF"` will not yield the results you might expect. The problem is caused by the zero (`0x00`) bytes in this string, which Silk Performer (like C) uses for string termination. Therefore all bytes after the first zero byte in a string are ignored when performing such string concatenation.

For manipulation of binary data that may contain zero bytes, you should use the function `SetMem`. Using the function library detailed below, the above request can be broken into the following lines:

```
ResetRequestData();
AppendRequestData("\h0000104f000000", 7);
AppendRequestData(sName); // e.g., "Peter"
AppendRequestData("\hFFFF");
SendTcpipRequest(hSend);
```

The function library is included using an include file (`*.bdh`). It uses two global variables for the request data and request data length. The function `AppendRequestData` appends a string (which may contain zero bytes) to the request buffer, and `SendTcpipRequest` sends the request data using an open TCP connection.

Here are the contents of the include file, which consists of three functions:

```
const
  BUFSIZE := 10000; // maximal size for request data
var
  // global variables for Request data contents and length
  gsRequestData : string(BUFSIZE);
  gnRequestLen  : number init 0;

dclfunc
  // start a new request string
  function ResetRequestData
  begin
    gnRequestLen := 0;
  end ResetRequestData;

  // append data (of length nLen) to the request string
  // if nLen=0, use strlen(sData) instead
  function AppendRequestData(sData: string;
                             nLen: number optional): number
  begin
    if nLen = 0 then nLen := strlen(sData); end;
    if nLen + gnRequestLen <= BUFSIZE then
      // append sData to gsRequestData
      SetMem(gsRequestData, gnRequestLen + 1, sData, nLen);
      // the request length has grown by nLen bytes:
      gnRequestLen := gnRequestLen + nLen;
    else
      RepMessage("Request buffer too small!",
                 SEVERITY_ERROR);
    end;
    AppendRequestData := gnRequestLen;
  end AppendRequestData;

  // Send the request buffer
  // (TCP-connection identified by hTcp)
  function SendTcpipRequest (hTcp: number): boolean
  begin
    SendTcpipRequest :=
      WebTcpipSendBin(hTcp, gsRequestData, gnRequestLen);
  end SendTcpipRequest;
```

Searching in Response Data

When zero bytes are contained in response data, it makes it difficult to search for strings. The reason is that the `StrSearch` and `StrSearchDelimited` functions search in a source string only until they hit the first zero byte, which is interpreted as the string terminator.

Silk Performer offers the function `BinSearch`, which works on binary data exactly as `StrSearch` does on strings. You can search for a string (or a sequence of binary data) in response data as follows:

```
nPos := BinSearch(sResponseData, nResponseLength, sSearch);
```

There is no binary counterpart for the `StrSearchDelimited` function. If you only want to search for strings (as opposed to binary data containing zero bytes), a workaround is to introduce a function that first eliminates all zero bytes from the response data by mapping them (for example, to the byte `0xFF`).

Here is a simple version of such a function. Note that it works only if `sLeftVal, sRightVal` and the string you are searching are strings (meaing, they don't contain zero bytes).

```
function BinSearchDelimited(sSource   : string;
                            nSrcLength: number;
                            sLeftVal  : string;
                            sRightVal : string)
                           :string(BUFSIZE)
var
  i : number;
begin
  // eliminate zero bytes in the source string
  for i:=1 to nSrcLength do
    if ord(sSource[i]) = 0 then
      sSource[i] := chr(255);
    end;
  end;
  StrSearchDelimited(BinSearchDelimited, BUFSIZE, sSource,
                     sLeftVal, 1, sRightVal, 1,
                     STR_SEARCH_FIRST);
end BinSearchDelimited;
```

Session IDs

Since many TCP/IP based protocols rely on TCP/IP connections between server and client to remain open and active during sessions, they don't contain session IDs. Therefore the problems that can arise when customizing test scripts for stateless protocols such as HTML often don't exist.

There are of course exceptions to this rule, so pay attention to session IDs. The TCP support in TrueLog Explorer can be helpful in this regard.

Finding Information

Standard protocols are typically defined in Requests for Comments (RFCs). You'll find samples of these at *http://www.rfc-editor.org/rfcxx00.html*. `[RFC854], [RFC2355]` and others are relevant to the examples included in the following topics.

Telnet, TN3270e, and Custom Protocols

The following topics apply the theories discussed in the preceding topics to specific real-world examples.

Telnet Protocol

The introduction of RFC 854, the Telnet protocol specification, states: "The purpose of the TELNET Protocol is to provide a general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to facilitate the interfacing of terminal devices and terminal-oriented processes."

A Telnet session consists of two main parts:

- A connection is established, and session details and options are negotiated between client and server.

- The application launches. From that point forward, generated traffic is user-driven.

The following Silk Performer Telnet script comes from a project in which a script was recorded and customized from a Telnet session with VT220 terminal emulation. The client (the terminal emulation software) is NT based and the server application resides on a Unix box.

Part I: Establishing a Connection

Here is the first part of the Telnet session recording:

```
WebTcpipConnect(hWeb0, "myserver", 23);

WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFC24", 3); // ·ü$

WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFB18", 3); // ·û·

WebTcpipRecvExact(hWeb0, NULL, 6);
WebTcpipSendBin(hWeb0, "\hFFFA18005654323230FFF0", 11);
                                   // ·ú··VT220·ð

WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFC20", 3); // ·ü

WebTcpipRecvExact(hWeb0, NULL, 60);
```

There is only one readable string in the inline comments: `VT220`, the terminal type. This is a first hint at the semantics of this Telnet traffic; server and client are negotiating terminal type and various other session settings.

In terms of script customization, nothing needs to be changed here. Silk Performer virtual users should negotiate session settings exactly as the terminal emulation software has done here.

Nevertheless, an analysis of the `record.log` file, along with the information from RFC 854 (the Telnet protocol specification), reveals how the conversation was achieved. The first part of the log is translated into TELNET codes in the following table:

Server-to-Client	Client-to-Server
FF FD 24 *(IAC DO ENVIRONMENT VARIABLES)*	
	FF FC 24 *(IAC WON'T ENVIRONMENT VARIABLES)*
FF FD 18 *(IAC DO TERMINAL-TYPE)*	
	FF FB 18 *(IAC WILL TERMINAL-TYPE)*
FF FA 18 01 FF F0 *(IAC SB TERMINAL-TYPE SEND IAC SE)*	
	FF FA 18 00 56 54 32 32 30 FF F0 *(IAC SB TERMINAL-TYPE IS "VT220" IAC SE)*
FF FD 20 *(IAC DO TERMINAL-SPEED)*	
	FF FC 20 *(IAC WON'T TERMINAL-SPEED)*
...	...

Each command begins with the "Interpret as Command" (IAC) escape character `0xFF`. Server and client agree on a terminal type (`VT220`) and a number of other session settings. Telnet session settings that can be negotiated between client and server include terminal speed, echo, "suppress go ahead," window size, remote flow control, and more.

Part II: User Interaction

The second part of the script contains the user interaction. The script generated from the recording session looks like this:

```
// …
    // login: send username
    WebTcpipSend(hWeb0, "t");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "e");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "s");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "t");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "u");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "se");
    WebTcpipRecvExact(hWeb0, NULL, 2);
    WebTcpipSend(hWeb0, "r");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "\r");
    WebTcpipRecvExact(hWeb0, NULL, 2);
    WebTcpipRecvExact(hWeb0, NULL, 10);
    // login: send password
    WebTcpipSend(hWeb0, "password\r");
    WebTcpipRecvExact(hWeb0, NULL, 2);
    WebTcpipRecvExact(hWeb0, NULL, 230);
    WebTcpipRecvExact(hWeb0, NULL, 47);
    WebTcpipRecvExact(hWeb0, NULL, 58);
    WebTcpipRecvExact(hWeb0, NULL, 40);
    WebTcpipRecvExact(hWeb0, NULL, 594);
    WebTcpipRecvExact(hWeb0, NULL, 340);
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipRecvExact(hWeb0, NULL, 188);
    WebTcpipRecvExact(hWeb0, NULL, 147);
    // Choose "1" from main menu and hit RETURN
    ThinkTime(7.0);
    WebTcpipSend(hWeb0, "1");
    WebTcpipRecvExact(hWeb0, NULL, 1);
    WebTcpipSend(hWeb0, "\r");
    WebTcpipRecvExact(hWeb0, NULL, 2);
    WebTcpipRecvExact(hWeb0, NULL, 7);
    WebTcpipRecvExact(hWeb0, NULL, 21);
    WebTcpipRecvExact(hWeb0, NULL, 691);
    // …
```

As you can see from the inline comments, this part of the script contains a login process (account name and password), followed by the selection of an item from a main menu (by hitting the `1` and `<RETURN>` keys).

If you leave a recorded script unchanged, it will typically play back without problems. However changes do have to be applied to scripts for data parameterization, and response verification.

Each keystroke is sent to the server as a single byte (without header or footer). The log file reveals that the server sends back the same byte as an echo:

```
WebTcpipSend(hWeb0, "s");
WebTcpipRecvExact(hWeb0, NULL, 1);
TcpipServerToClient(#432, 1 bytes)
{
  s
}
```

Looking back at the recorded script, you'll find that the sending of the password, as opposed to the sending of the user login name, doesn't trigger a sequence of echoes. This is because the password isn't supposed to appear on the terminal screen.

Note that the communication is full duplex. This means that both server and client can send simultaneously; they don't have to wait for each other. In the above example, you can see the result of this in the lines:

```
WebTcpipSend(hWeb0, "se");
WebTcpipRecvExact(hWeb0, NULL, 2);
```

Here, because of rapid typing during the recording session, the echo `s` came back only after the `e` keystroke had been sent to the server.

Once the password is sent, a number of `WebTcpipRecvExact` statements follow in the above code. In the recording's log file, one of these statements looks like this:

```
TcpipServerToClientBin(#432, 59 bytes)
{
  00000000 [8;19H·[;7m+---  1B 5B 38 3B 31 39 48 1B 5B 3B 37 6D 2B 2D 2D 2D
  00000010 ---Hinweis: Kein 2D 2D 2D 48 69 6E 77 65 69 73 3A 20 4B 65 69 6E
  00000020 Kunde gefunden-  20 4B 75 6E 64 65 20 67 65 66 75 6E 64 65 6E 2D
  00000030 ------+·[m·      2D 2D 2D 2D 2D 2D 2B 1B 5B 6D 0A
}
```

The server response contains plain text as well as meta information, such as text position and format (note that in German, "Hinweis: Kein Kunde gefunden" means "Message: No customer found").

In this Telnet example, determining when the server response is complete is challenging. First, the packet length is not included in the response data (Case I). Second, there is no termination byte sequence (Case II). Therefore this example represents Case III: No information on response packet size from section “Dynamically Receiving Server Responses”.

Note: In other Telnet based projects, you may find terminator strings in server response data (for example, the command prompt character). This depends entirely on the application under test.

To solve this problem generically, a TelnetReceive-Response function that accepts incoming server responses of unspecified length in a loop is written. The loop is terminated when the client waits for a new response packet for more than a specified number of seconds. The corresponding function code is included later in this section.

Looking at the same part of the script after customization, the structure is more visible, and the server responses are handled by the new function. Note that complete strings can be sent to the server, as opposed to sending each key stroke as a single packet. You consequently don't have to wait for each echo character individually, because they can be read asynchronously (due to the full-duplex nature of the Telnet protocol) after sending the complete request.

```
// Login: Send Username
WebTcpipSend(hWeb0, "testuser\r");
TelnetReceiveResponse(hWeb0, 1.0, "Login: Username");

// Login: Send Password
WebTcpipSend(hWeb0, "password\r");
TelnetReceiveResponse(hWeb0, 5.0, "Login: Passwort");

// Choose "1" from main menu and hit RETURN
WebTcpipSend(hWeb0, "1\r");
TelnetReceiveResponse(hWeb0, 5.0, "Choose 1 from menu");
```

The `TelnetReceiveResponse` function eliminates the need to wait for incoming server data for appropriate periods of time. It takes three parameters:

`hWeb0`: Is the handle of the open TCP connection `fTimeout`: Defines how to decide when the server response is complete: If after `fTimeout` seconds, no further server response is available, the function returns.

`sAction`: This string is used for appropriate naming of the custom timer, and can be used for logging and debugging purposes.

Example

```
function TelnetReceiveResponse(hWeb0: number;
                               fTimeout: float;
                               sAction: string): number
var
  sData:      string(4096);
  nRecv:      number;
  nRecvSum:   number;
  fTime:      float;

begin
  gsResponse := "";
  nRecvSum   := 0;

  MeasureStart(sAction);

  while WebTcpipSelect(hWeb0, fTimeout) do
    if NOT WebTcpipRecv(hWeb0, sData, sizeof(sData), nRecv)
    then
      exit;
    end;
    if nRecv = 0 then
      exit;
    end;
    SetMem(gsResponse, nRecvSum + 1, sData, nRecv);
    nRecvSum   := nRecvSum + nRecv;
  end;

  MeasureStop(sAction);
  MeasureGet(sAction, MEASURE_TIMER_RESPONSETIME,
             MEASURE_KIND_LAST, fTime);

  if fTime > fTimeout then
    fTime := fTime - fTimeout;
  end;
  MeasureIncFloat("RespTime: " + sAction, fTime, "sec",
                  MEASURE_USAGE_TIMER);

  TelnetReceiveResponse := nRecvSum;

end TelnetReceiveResponse;
```

The function works as follows: It waits until a server response is ready to be read in under `fTimeout` seconds (`WebTcpipSelect`). As soon as a server response is available, it is received and appended to the global string variable `gsResponse`. The loop terminates when the server response is empty, when the timeout is exceeded, or if `WebTcpipRecv` fails for any reason.

This loop structure is necessary because often a Telnet server will send a line of characters, nothing will happen for a couple of seconds, and then suddenly more lines come in.

Some care must be taken when looking at these time measurements. Because of the nature of the timeout, the time measurements usually include a final timeout period of `fTimeout` seconds. This has to be subtracted from the measured time to get the true roundtrip time measurement. This corrected time measurement is made available as a custom measurement with the name `"RespTime: " + sAction` and the dimension seconds.

TN3270e Protocol

The TN3270e protocol is a method of emulating 3270 terminal and printer devices using Telnet. It is used by terminal emulation software such as Rumba® and Hummingbird's HostExplorer® for direct connections to mainframes.

Similar to Telnet, a typical TN3270e session consists of two main parts:

- A connection is established; session details and options are negotiated.
- The application starts, and from that point forward generated traffic is user-driven.

Part I: Establishing a Connection

The first part of a typical recorded script of a TN3270e session looks like this:

```
WebTcpipConnect(hWeb0, "10.19.111.201", 7230);
WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFB28", 3);    // ·û(
WebTcpipRecvExact(hWeb0, NULL, 7);
WebTcpipSendBin(hWeb0,
  "\hFFFA28020749424D2D333237382D342D"
    // ·ú(··IBM-3278-4-
  "\h45FFF0", 19);                        // E·ð
WebTcpipRecvExact(hWeb0, NULL, 28);
WebTcpipSendBin(hWeb0, "\hFFFA2803070004FFF0", 9);
  // ·ú(·····ð
WebTcpipRecvExact(hWeb0, NULL, 9);
WebTcpipRecvExact(hWeb0, NULL, 347);
```

This looks similar to the first part of the Telnet session from the previous section because the TN3270e protocol is based on the Telnet protocol. Again, nothing needs to be changed here. Because this is a stateful, connection-oriented protocol, session handling is not an issue.

The traffic from the first part of the session is translated into TELNET code in the following table:

Server-to-Client	Client-to-Server
FF FD 28 (IAC DO TN3270E)	
	FF FB 28 (IAC WILL TN3270E)
FF FA 28 08 02 FF F0 (IAC SB TN3270E SEND DEVICE-TYPE IAC SE)	
	FF FA 28 02 07 49 42 4D 2D 33 32 37 38 2D 34 2D 45 FF F0 (IAC SB TN3270E DEVICE-TYPE REQUEST "IBM-3278-4-E" IAC SE)
FF FA 28 02 04 49 42 4D 2D 33 32 37 38 2D 34 2D 45 01 54 39 35 49 54 51 4D 55 FF F0 (IAC SB TN3270E DEVICE-TYPE IS "IBM-3278-4-E" CONNECT "T95ITQMU" IAC SE)	
	FF FA 28 03 07 00 04 FF F0 (IAC SB TN3270E FUNCTIONS REQUEST [BIND-IMAGE SYSREQ] IAC SE)
FF FA 28 03 04 00 04 FF F0 (IAC SB TN3270E FUNCTIONS IS [BIND-IMAGE SYSREQ] IAC SE)	

In summary, server and client agree on a protocol, a device type (IBM-3278-4-E), and on whether or not to use certain protocol features. Note that the terminal name (`T95ITQMU`) is assigned by the server, not the client. This is because the server system holds a database that handles the mapping of client IP addresses to terminal names.

Part II: User Interaction

The second part of the script contains the user interaction. The following request-response pair example represents an interaction where the user enters an account number (`238729`) into a text field and then hits the `RETURN` key:

```
WebTcpipSendBin(hWeb0,
  "\h00000000007DC6C61140C4F311C640F2"
     // ·····}ÆÆ·@Äó·Æ@ò
  "\hF3F8F7F2F9FFEF", 23);              // óø÷òù·ï
WebTcpipRecvExact(hWeb0, NULL, 199);
```

Knowing that the traffic is encoded in EBCDIC (rather than ASCII), the account number can be found in the request string using an EBCDIC code table: `238729` is represented as the binary byte sequence `0xF2 F3 F8 F7 F2 F9` in EBCDIC. This is the part that is relevant to customization; the rest may be left unchanged.

Here's an account of the other parts of this message. The first five bytes represent the TN3270e message header (`RFC 2355`):

Field	Length	Value in our example	
Data type	1 byte	0x00	3270-DATA
Request flag	1 byte	0x00	ERR-COND-CLEARED
Response flag	1 byte	0x00	NO-RESPONSE
Sequence number	2 bytes	0x0000	Sequence numbers may or may not be used. In this case, they aren't.

Each client request is terminated by a two-byte sequence: `0xFF EF`.

The remainder of the request data (bytes #6 - #21 in this example) is application data containing screen positions, key codes, and text.

The server response can be analyzed in the corresponding log or TrueLog files from the recording session. In this example, the response is a 199-byte data block beginning with five zero bytes (`0x00`) and ending with `0xFF EF`, just like the request data. The data content in between is similar to the request data: cursor positions, formatting, and text content (encoded in EBCDIC).

A simple customization of the script extract above using the function library would look like this:

```
ResetRequestData();
AppendRequestData("\h00000000007DC6C61140C4F311C640",
                15);
AppendRequestData(ASC2EBCDIC(sAccountNr) + "\hFFEF");
SendTcpipRequest(hWeb0);
WebTcpipRecvUntil(hWeb, sResp, sizeof(sResp), nRecv,
                  "\hFFEF", 2);
```

Here a string variable `sAccountNr` has been introduced for the account number (where `238729` was used during recording). The function `ASC2EBCDIC` that converts between ASCII and EBCDIC is explained in the following section.

Finally, the inflexible `WebTcpipRecvExact` has been replaced with `WebTcpipRecvUntil`, which is appropriate here because the trailing byte sequence is known. The response data is stored in the string variable `sResp`, and `nRecv` contains the number of received bytes.

For logging and verification purposes, the server response should be translated from EBCDIC to ASCII. For example:

```
Writeln("Response: " + EBCDIC2ASC(sResp, nRecv));
```

When necessary, single response data (such as a new ID that's needed as input data for subsequent requests) can be extracted from this response using the `StrSearchDelimited` function.

Taking customization a step further, it's good practice to replace the `WebTcpipRecvUntil` call in the script above with a `MyWebTcpipRecv` function that encapsulates all the necessary actions on server responses:

- Call `WebTcpipRecvUntil` with the appropriate end byte sequence.
- Convert the response from EBCDIC to ASCII.
- Log the response to the output file (in ASCII).
- Do generic error checking on the response by searching for common error messages.
- Make the response data available as a return value or global variable for further verification or data extraction.

In the Silk Performer script, each client request should be followed by a call to this new function, replacing the `WebTcpipRecvExact` function calls from the recording session.

EBCDIC to ASCII Character Code Conversion

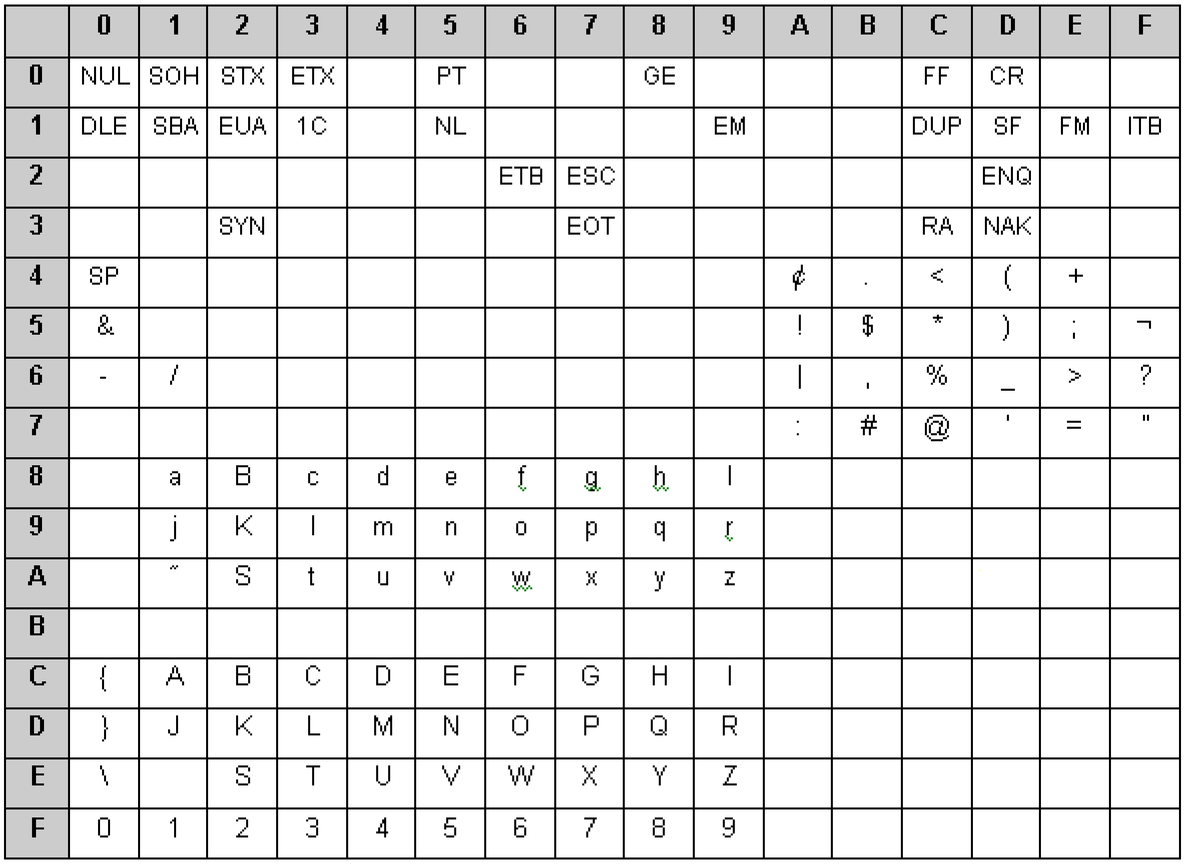
EBCDIC (Extended Binary Coded Decimal Interchange Code) is the data alphabet used in all IBM computers, except personal computers. A conversion routine that translates server responses from the EBCDIC character set to ASCII is easy to implement.

The function `EBCDIC2ASC` uses the code map defined by the array as `EBCDIC_2_ASCII`, which maps each EBCDIC character to its ASCII equivalent. The function `ASC2EBCDIC`, which is required for translating client requests from ASCII to EBCDIC, works in a similar manner.

```
var
  asEBCDIC_2_ASCII : array[256] of string(1) INIT
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", "a", "b", "c", "d", "e", "f", "g", "h", "i", " ", " ", " ", " ", " ",
" ",
" ", "j", "k", "l", "m", "n", "o", "p", "q", "r", " ", " ", " ", " ", " ",
" ",
" ", "~", "s", "t", "u", "v", "w", "x", "y", "z", " ", " ", " ", " ", " ",
" ",
" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ",
" ", "A", "B", "C", "D", "E", "F", "G", "H", "I", " ", " ", " ", " ", " ",
" ",
" ", "J", "K", "L", "M", "N", "O", "P", "Q", "R", " ", " ", " ", " ", " ",
" ",
" ", " ", "S", "T", "U", "V", "W", "X", "Y", "Z", " ", " ", " ", " ", " ",
" ",
"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", " ", " ", " ", " ", " ",
" ";

function EBCDIC2ASC(pEBCDIC: string; pMaxLen: number optional)
  var
    i: number;
  begin
  //
  if pMaxLen = 0 then
     pMaxLen := Strlen(pEBCDIC);
  // writeln("length of string : " + string(pMaxLen));
  end;
  //
  // writeln("EBCDIC STRING " + pEBCDIC);
  //
  for i := 1 to pMaxLen do
  //
  //writeln("Ordinal Value:" + String(ord(pEBCDIC[i])));
  //write("Value From Array:" + asASCII_2_EBCDIC[ord(pEBCDIC[i]) + 1]);
    write(asEBCDIC_2_ASCII[ord(pEBCDIC[i]) + 1]);
  //writeln;
  //
  end;
    writeln;
  end EBCDIC2ASC;
```

The table below is the standard EBCDIC table for the 2780/3780 Protocol code map (taken from [EBCDIC_CTI]). As an example, to decode the EBCDIC byte `0x83`, choose row 8 and column 3. You'll find that `0x83` maps to the letter c in ASCII.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX		PT			GE				FF	CR		
1	DLE	SBA	EUA	1C		NL				EM			DUP	SF	FM	ITB
2							ETB	ESC						ENQ		
3			SYN					EOT					RA	NAK		
4	SP										¢	.	<	(	+	
5	&										!	$	*	)	;	¬
6	-	/									|	,	%	_	>	?
7											:	#	@	'	=	"
8		a	B	c	d	e	f	g	h	I						
9		j	K	l	m	n	o	p	q	r						
A		˜	S	t	u	v	w	x	y	z						
B																
C	{	A	B	C	D	E	F	G	H	I						
D	}	J	K	L	M	N	O	P	Q	R						
E	\		S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						

[EBCDIC_UNI] is a reference that presents the specifications of the UTF-EBCDIC - EBCDIC Friendly Unicode (or UCS) Transformation Format.

Issues with "Keep Alive" Mechanisms

The following test script sequence was taken from a TN3270e traffic recording:

```
WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFB06", 3);  // IAC WILL TIMING-MARK
WebTcpipRecvExact(hWeb0, NULL, 3);
WebTcpipSendBin(hWeb0, "\hFFFC06", 3);  // IAC WON'T TIMING-MARK
```

In this example, the server challenges the client with a three-byte code (first line of the code above) whenever the client is inactive for a period of time. The client responds with three-byte sequence of its own. Such "ping pong" activity, initiated by the server, can serve as a control mechanism for detecting whether or not a client is still active.

These lines cause problems during replay because they aren't predictable or reproducible. To address such a situation, rather than incorporating the appropriate intelligence into a Silk Performer script, which would be a daunting task, simply disable this feature on the server.

Some background info from RFC 860 (Telnet timing mark option):

"It is sometimes useful for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise disposed of. This option provides a mechanism for doing this. In addition, even if the option request (`DO TIMING-MARK`) is refused (by `WON'T TIMING-MARK`) the requester is at least assured that the refuser has received (if not processed) all previous data."

IP Spoofing

For successful replay of such a script with parallel virtual users, each user must use a unique IP address.

Enable the **Client IP address multiplexing** option in Silk Performer at **Settings** > **Active Profile** > **Internet** > **Optimization** and configure enough IP addresses on Silk Performer agents so that each virtual user can have a unique IP address ("IP Spoofing"). IP addresses can also be configured using the Silk Performer System Configuration Manager at **Tools** > **System Configuration Manager** > **IP Address Manager**.

Recording Rule Configuration Files

The Silk Performer recorder can be configured to generate correct `WebTcpipRecvUntil` calls for the TN3270e protocol automatically using a recording rule configuration file.

Two different rules must be specified: in the first part of the session where the session parameters are negotiated, `0xFF F0` is used as the termination sequence, while `0xFF EF` serves as the termination sequence in the 3270-DATA part of the session.

These two parts can be distinguished by checking the first byte of the response data. In the first part, it is equal to `0xFF` (Telnet code IAC - "Interpret as Command"), while it is equal to `0x00` (Code `3270-DATA`, compared to the TN3270e response header above) in the second part.

The resulting recording rule XML file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordingRuleSet>
  <TcpRuleRecvUntil>
    <Name>Telnet session start</Name>
    <Identify>
      <TermData>&#xFF;&#xF0;</TermData>
      <IgnoreWhiteSpaces>false</IgnoreWhiteSpaces>
    </Identify>
    <Conditions>
      <CompareData>
        <Data>&#xFF;</Data>
        <ApplyTo>Left</ApplyTo>
        <Offset>0</Offset>
      </CompareData>
    </Conditions>
  </TcpRuleRecvUntil>
  <TcpRuleRecvUntil>
    <Name>3270-DATA</Name>
    <Identify>
      <TermData>&#xFF;&#xEF;</TermData>
      <IgnoreWhiteSpaces>false</IgnoreWhiteSpaces>
    </Identify>
    <Conditions>
      <CompareData>
        <Data>&#x00;</Data>
        <ApplyTo>Left</ApplyTo>
        <Offset>0</Offset>
      </CompareData>
    </Conditions>
  </TcpRuleRecvUntil>
</RecordingRuleSet>
```

Custom TCP/IP Based Protocols

The following example comes from an application that used an entirely custom TCP/IP based protocol. In this protocol, both the client request and server response data obey a set of predefined message telegram structures consisting of fixed-length fields. Some of the fields are binary (and may therefore contain zero bytes) and some are string type.

Here is a typical request-response pair from the recorded traffic:

```
WebTcpipConnect(hWeb0, "10.9.96.81", 3311);
WebTcpipSendBin(hWeb0,
"\h00000280000000000000000000000000" // ················
"\h00010000000000000000004F4B36302020" // ············OK60
...
"\h303030302B303030303030303030303030" // 0000+00000000000
"\h30303030000000000000000000000000" // 0000············
, 640);
WebTcpipRecvExact(hWeb0, NULL, 80);
WebTcpipShutdown(hWeb0);
```

Here is the server response from the record log. Based on the third argument of the `WebTcpipRecvExact` function in the above script, we already know that it contains 80 bytes:

```
00 00 00 00  ···P············  00 00 00 50 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 10  ············OK60  00 01 00 00 00 00 00 00 00 00 4F 4B 36 30 20 20
00 00 00 20    ··U60OKS1 SILK  20 20 00 00 55 36 30 4F 4B 53 31 20 53 49 4C 4B
00 00 00 30  01  ············  30 31 20 20 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 40  ················  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Note: The first four bytes (`0x00 00 00 50`) are a representation of a four-byte integer (a long variable) with the decimal value `80 (5 * 16 = 80)`. Note that the length of the entire packet is 80 bytes, hence the 4 initial bytes are included in the "packet size" variable. If available, consult protocol documentation regarding the custom application under test. Otherwise you will have to experiment to discover such protocol behavior on your own.

Note: The client request telegram contains the same information. The first four bytes are `0x00 00 02 80`, equal to `2*256 + 8*16 = 640`. This is the number of bytes sent to the server.

Here is the script after customization. Note that:

- An added pair of `MeasureStart`/`MeasureStop` functions has been added to measure the time for the request.
- The request has been split into several pieces for parameterization, using the function library.
- The `WebTcpipRecvExact` function has been replaced by a call to `WebTcpipRecvProto`.

Here is the result:

```
ResetRequestData();
AppendRequestData("\h000002800000000000000000000000"
                 "\h00010000000000000000", 26);
AppendRequestData("OK60    ");
AppendRequestData("\h0000", 2);
AppendRequestData(sUser + sPassword);
// ... (some more lines not displayed here)

MeasureStart("Write Journal");
WebTcpipConnect(hWeb0, "10.9.96.81", 3311);
SendTcpipRequest(hWeb0);
WebTcpipRecvProto(hWeb0, 0, 4, TCP_FLAG_INCLUDE_PROTO,
               sResponse, sizeof(sResponse), nReceived);
// ... (response verification not displayed here)
WebTcpipShutdown(hWeb0);
MeasureStop("Write Journal");
```

Recording Rule Configuration Files

The following recording rule XML file configures the Silk Performer recorder so that it generates the correct `WebTcpipRecvProto` calls in place of `WebTcpipRecvExact`. The settings in the `Identify` node map to the arguments of `WebTcpipRecvProto` in the script extract above.

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordingRuleSet>
  <TcpRuleRecvProto>
    <Name>My custom TCP protocol</Name>
    <Identify>
      <LengthOffset>0</LengthOffset>
      <LengthLen>4</LengthLen>
      <OptionFlags>ProtoIncluded</OptionFlags>
    </Identify>
  </TcpRuleRecvProto>
</RecordingRuleSet>
```

Recorder Settings

Silk Performer offers two mechanisms for recording TCP/IP based protocol traffic. Socksifying such an application is the most commonly accepted approach. There are however some scenarios in which using a TCP proxy recorder is the better solution:

- Using the TCP proxy mechanism, Silk Performer doesn't have to run on the same machine as the client application. This works if for example both client and server are Unix applications. Just configure them in such a way that they connect to each other using the Silk Performer proxy residing on a Windows box.
- Sometimes, there are numerous client application processes all connected to the same server. In such instances it may be quicker to set up a TCP proxy than to it is to search NT's task manager for all processes that are generating traffic.
- Configuration is easy if you only need to record on one or a small number of TCP ports.

"Socksifying" an Application

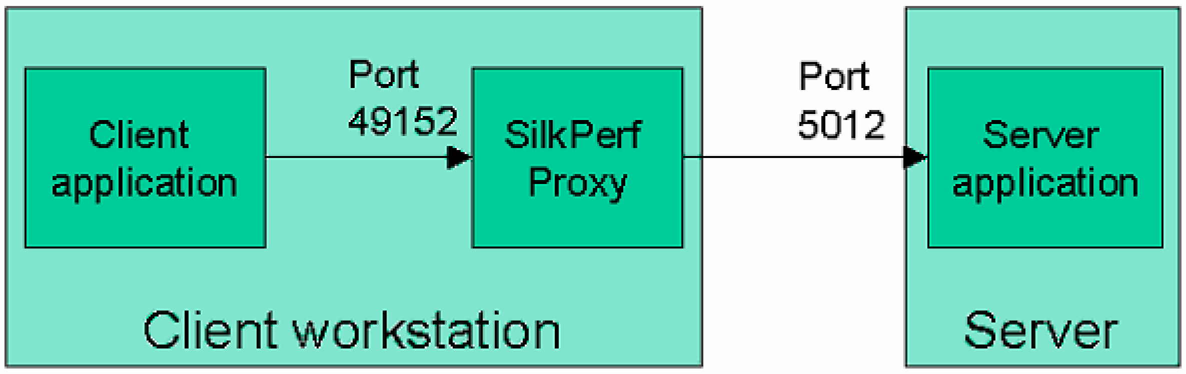
If you know the path of the client application executable that's connecting to the server, you can set up a recording profile for the application in Silk Performer at **Settings** > **System** > **Recorder** > **Recording Profiles** > **Add**.

TCP Proxy Recorder

An alternative solution involves using a TCP/IP proxy recorder. As a prerequisite, you must be able to control the server name and TCP port number the client application connects to. Typically, these settings can be found in the registry or in a configuration file. Then, you can configure Silk Performer and the client application so that traffic is explicitly routed over the Silk Performer recorder.

Assuming that the client connects to `MYSERVER` on port `5012`, you would change these settings to `LOCALHOST` and port `49152` (or any other unused TCP port) in the registry or appropriate configuration file. In Silk Performer, choose **Settings** > **System** > **Recorder** > **Proxies** > **Add**. Add a new proxy, and configure it as a TCP proxy listening on port `49152`, connecting to the remote host `MYSERVER` on port `5012`.

In this way, the following scenario is configured:



The client application doesn't notice a difference in performance when the recorder is running. This is in contrast to the scenario in which the client is connected directly to the server.

Load Testing PeopleSoft

This section explains how to use Silk Performer to load test PeopleSoft applications. It explains how to use the PeopleSoft SilkEssential package to record and customize PeopleSoft scripts for realistic simulation of virtual users.

Specifying the PeopleSoft Project Type

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. In the **Type** menu tree, select **ERP/CRM** > **PeopleSoft**.
5. Click **Next** to create a project based on your settings.

Script Modeling

Script modeling for PeopleSoft transactions is straightforward. Recorded scripts work without any required modification, however by following a few standard customization steps after recording, you can get more out of recorded scripts. Recorded scripts are designed to make the process quick and easy.

Recording

Before recording, make sure that the project has been created based on the PeopleSoft project type. Record user interactions in a single BDL transaction. Ensure that the transaction begins with the sign-in process and ends with the sign-out process. Do not use the back button of your browser as this is generally problematic for highly state-dependent web applications such as PeopleSoft.

PeopleSoft scripts have a standard structure. Apart from the sign-in and sign-out processes, there are basically two types of user interaction involved:

- Navigation in the menu tree
- Work in the work area

Inclusion of PeopleSoft API Functions

The recorded script contains a use statement for `PeopleSoft8Api.bdh`, a BDH file that contains PeopleSoft specific API functions. It also contains `PeopleSoftInit()`, the initialization function call in the `TInit` transaction (see the example below).

This initializes the PeopleSoft framework contained in several BDH files of the PeopleSoft SilkEssential package. It also enables global verification rules to catch PeopleSoft specific application-level errors. This is especially useful when used in conjunction with the TrueLog On Error option.

In addition, it enables global parsing rules for the parsing of dynamic form names. Parsed form names are available in the global variable `gsFormMain`.

Example: Initializing PeopleSoft API functions

```
benchmark SilkPerformerRecorder

use "WebAPI.bdh"
use "PeopleSoftApi.bdh"

dcluser
  user
    VUser
  transactions
    TInit : begin;
    TMain : 1;

var
  // ...

dclrand

dcltrans
  transaction TInit
  begin
    WebSetBrowser(WEB_BROWSER_MSIE6);
    WebModifyHttpHeader("Accept-Language", "en-us");
    WebSetStandardHost("crm.ps.my.company.com");
    PeopleSoftInit();
    //GetLoginInfoPS("LoginPS.csv", gsUserId, gsPassword);
    //WebSetUserBehavior(WEB_USERBEHAVIOR_FIRST_TIME);
    //WebSetDocumentCache(true, WEB_CACHE_CHECK_SESSION);
  end TInit;
```

Wrapper Functions

The BDH files contain wrapper functions for all page-level API functions. These wrapper functions call the original API functions and perform some additional tasks.

The following wrapper functions are defined:

Wrapper function	Wrapped original function
`WebPageUrlPS`	`WebPageUrl`

Wrapper function	Wrapped original function
`WebPageLinkPS`	`WebPageLink`
`WebPageSubmitPS`	`WebPageSubmit`
`WebPageSubmitBinPS`	`WebPageSubmitBin`
`WebPageFormPS`	`WebPageForm`
`WebPageFileUploadPS`	`WebPageFileUpload`

Sign-In and Sign-Out

Recorded scripts should begin with the sign-in process and end with the sign-out process, as shown in the example below. The BDH files define and the recorder records, the functions `HomepagePS`, `SignInPS` and `SignOutPS`.

Example: Recorded sign-in and sign-out processes

```
var
  gsHomepageUrl : string init "http://standardhost/psp/ps/
?cmd=login";

transaction TMain
begin
  HomepagePS(gsHomepageUrl, "PeopleSoft 8 Sign-in");
  ThinkTime(6.0);
  SignInPS("login", LOGIN002, "EMPLOYEE"); // Form 1

  // more function calls ...
  SignOutPS(gsSignoutUrl, "PeopleSoft 8 Sign-in");
end TMain;
```

These function calls do not require customization. the `SignInPS` function parses the sign-out URL to the `gsSignoutUrl` variable, and the recorded script uses the variable in the `SignOutPS` function call. The recorder records the URL of the homepage into a variable for easy customization.

Navigation in the Menu Tree

Navigation in the menu tree is recorded by a `WebPageLinkPS` call that uses a custom hyperlink from a `WebPageParseUrl` call of a previous API call.

Example: Recorded navigation in the menu tree

This process does not require any modification:

```
WebPageParseUrl("JavaScript Link in page EMPLOYEE",
                "DEFINITION\",\"", "\"",
                 WEB_FLAG_IGNORE_WHITE_SPACE);
WebPageLinkPS("Home", "EMPLOYEE"); // Link 1

WebPageLinkPS("JavaScript Link in page EMPLOYEE",
              "CR_PRODUCT_DEFINITION", 3);
```

Interaction in the Work Area

User interaction in the work area is recorded by a `WebPageSubmitPS` call. The form relies on the form field attribute `USE_HTML_VAL`, and thus ensures proper context management without customization.

The form name may change dynamically. While the form name is usually `main` or `win` (depending on the PeopleSoft version), it may, depending on server load, become `main1`, `main2`, `win1`, `win2`, and so on.

Because of this, the recorder records a `WebPageSubmitPS(NULL, ...)` function call that references the form by ordinal number rather than form name.

However, the BDH files implement global parsing for the dynamic form name during script execution, so the actual form name of the current page is always available in the global variable `gsFormMain`.

Example: User interaction within the work area

```
WebPageSubmitPS(NULL, MAIN005, "Product Definition", 4); // Form 4
```

The above example shows a `WebPageSubmitPS` function, which references the dynamic form by its ordinal number on the page.

The next example shows a typical submitted form. There is no need for customization of session or state management here because of the `USE_HTML_VAL` attributes. The only customization that may be required is the randomization of input values.

Example: Submitted form

```
MAIN008:
"ICType"       := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "Panel"
"ICElementNum" := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "0"
"ICStateNum"   := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "5"
"ICAction"     := "#ICSearch",
               // hidden, changed(!)
"ICXPos"       := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "0"
"ICYPos"       := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "0"
"ICFocus"      := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: ""
"ICChanged"    := "" <USE_HTML_VAL> ,
               // hidden, unchanged, value: "-1"
"RBF_PRD_DF_SRCH_SETID"      := "" <USE_HTML_VAL> ,
                             // unchanged, value: ""
"RBF_PRD_DF_SRCH_PRODUCT_ID" := "" <USE_HTML_VAL> ;
                             // unchanged, value: "NEXT"
```

Script Customization

The PeopleSoft SilkEssential package contains some utility functions that can be accessed through manual script enhancement to get more from recorded scripts. This section describes these optional steps.

Customizing Think Times

The recorder of Silk Performer records think times as they occur during recording. Often however, this is not what is required. The `PeopleSoftApi.bdh` file provides a substitute for the `ThinkTime` API function that is called `ThinkTimePS`.

```

function ThinkTimePS( fTime           : float optional;
                      bForceThinkTime : boolean optional);
```

To customize the script, click **Edit** in the menu and click **Replace**. In the **Replace** dialog box, replace all `ThinkTime` function calls with `ThinkTimePS` function calls.

The behaviour of the `ThinkTimePS` function depends on the value of the `bForceThinkTime` parameter. When `bForceThinkTime` is set to false (the default value), `ThinkTimePS` ignores the value of the `fTime` parameter and calls the `ThinkTime` API function with the value of the `ThinkTimePS` project attribute. The behaviour follows the usual thinktime-related profile settings (for example stresstest, random think time, exponential/uniform distribution, think time limited to x seconds, and so on).

When `bForceThinkTime` is set to true, `ThinkTimePS` calls the `ThinkTime` API function with the value of the parameter `fTime` and the `OPT_THINKTIME_EXACT` option. This results in a think time that is exactly as specified, regardless of profile settings and project attributes.

There are two project attributes available for customizing the behaviour of the `ThinkTimePS` function. Click **Project** in the menu and click **Project Attributes** to access these attributes:

- ThinkTimePS (float): Think time value used by `ThinkTimePS` when `bForceThinkTime` is false. Unit: Seconds; Preset: 10.0
- ForceThinkTime (boolean): Allows for global override of the `bForceThinkTime` parameter of the `ThinkTimePS` function. Preset: false

Uniquely Seed Randomness

To improve the randomness of script execution, it can be advantageous to uniquely seed the random number generator for each virtual user. This is done by changing the value of the `UniquelySeedRandomness` project attribute to true.

Customizing Timer Names

For easier results analysis, it is desirable to have timer names that can be sorted based on their occurrence in scripts and amended with additional information. To accomplish this, the wrapper functions amend original timer names with additional information.

Project attributes can be used to include any combination of the following items in amended timer names:

- Agent name
- Transaction name
- Modem emulation speed
- Page counter for the current transaction

The `WebPage*PS` wrapper functions keep track of internal page numbers, which are prepended to timer names when this option is selected in the project attributes. While page numbers are automatically reset to 0 at the end of each transaction, you can manually reset the page counter at any time by calling the `ResetPageCount` function.

You may also call the `IncPageCount` function to manually increment the internal page counter. This is useful for keeping a consistent page counter after an unbalanced if statement.

The `WebPage*PS` wrapper functions also allow you to specify a specific value for the page counter. This value may be passed as the parameter where the frame name is passed in the original wrapped functions.

Note: Wrapper functions do not allow you to pass a frame name. This rule is also obeyed by the recorder. You may pass the special value `PAGE_NUMBER_KEEP` to reuse the current page counter, rather than increment it.

Targeting a Different Server

Recorded scripts contain a call to `WebSetStandardHost` in the `TInit` transaction.

Example: Call to WebSetStandardHost as recorded

```
transaction TInit
begin
  // ...
  WebSetStandardHost("crm.ps.my.company.com");
  // ...
  PeopleSoftInit();
end TInit;
```

There are three options for targeting a different server:

1. Edit the recorded `WebSetStandardHost` function call.
2. Delete the recorded `WebSetStandardHost` function call and specify a standard host in the profile settings.
3. Specify a different standardhost in project attributes. These project attributes will be evaluated in the `PeopleSoftInit()` function call and will therefore override the recorded `WebSetStandardHost` function call.

Enable Server-Side Tracing

PeopleSoft offers the option to enable detailed server-side tracing for individual login session. Scripts should be recorded without tracing being enabled, because this would enable tracing for all virtual users.

Instead, server-side tracing can be enabled on a virtual user basis by inserting a call to the `EnableTracingPS` function in the `TInit` transaction, as shown in the sample below. The form used for login must be passed to this function. It is important to note that tracing should only be enabled for individual virtual users. Enabling tracing for all virtual users would impose significant overhead on servers and could skew test results.

Enabling tracing for one virtual user

```
transaction TInit
begin
  // ...
  PeopleSoftInit();
  if GetUserId() = 1 then
     EnableTracingPS(LOGIN001);
  end;
end TInit;

dclform
  LOGIN001:
    "httpPort" := "" <USE_HTML_VAL> ,
    "timezoneOffset" := "-60",
    "userid" := gsUserId,
    "pwd" := gsPassword,
    "Submit" := "" <USE_HTML_VAL> ;
```

Tracing options can be modified by editing the `Tracing.csv` file, which is available in the `Data Files` node of the project tree view.

Randomizing Table Row Selection

PeopleSoft pages often contain tables, for example lists of search results. Clicking an item within such a table returns a form where the **ICAction** field is set to `#ICRowX`, where `X` denotes the ordinal number of the selected row.

Example: Form submitted while selecting an item in a table

```
dclform
  MAIN003:
    "ICType" := "" <USE_HTML_VAL> ,
    "ICElementNum" := "" <USE_HTML_VAL> ,
    "ICStateNum" := "" <USE_HTML_VAL> ,
       "ICAction" := "#ICRow9", // hidden, changed(!)
       "ICXPos" := "" <USE_HTML_VAL> ,
       "ICYPos" := "" <USE_HTML_VAL> ,
       "ICFocus" := "" <USE_HTML_VAL> ,
       "ICChanged" := "" <USE_HTML_VAL> ,
// ...
```

To accurately randomize or customize such forms, you must determine the number of table rows.

There are four functions that assist with this:

- `GetMaxRowNr` returns the maximum valid row number (for example in a table with 47 rows the maximum valid row number is 46, since row numbers are zero-based). If the current page does not contain a table, it returns -1.
- `GetRowCount` returns the number of rows on the current page. If the current page does not contain a table, it returns 0.
- `GetRndRowStr` returns a valid, random row string.
- `FindICRow(sStringToFind : string) : string` returns the row string of the first row that contains the given text.

In the example below the `GetRndRowStr` function is used to randomize the selection of an item in a table.

Example: Randomizing the selection of an item in a table

```
dclform
  MAIN003:
// ..
    "ICStateNum" := "" <USE_HTML_VAL> ,
    "ICAction" := GetRndRowStr(),
    "ICXPos" := "" <USE_HTML_VAL> ,
// ...
```

Application-Level Errors

PeopleSoft does not use HTTP response status codes to indicate application-level errors. Instead, it returns HTML with status code `200 Success`, even when the HTML contains error messages.

There are two error message types:

- Error messages embedded in HTML.
- Error messages in the parameters of JavaScript functions (`Alert`) that display dialog boxes.

Additionally, not all messages that are displayed with an `Alert` function are error messages. Some messages are simply informational, for example: `Record has been saved`.

If severe errors occur, transactions should not be continued.

Using the PeopleSoft SilkEssential package, recorded scripts are automatically able to handle application-level errors. When severe errors occur, errors of `SEVERITY_TRANS_EXIT` severity are raised and virtual users gracefully terminate their transactions by signing out. It is strongly recommended to enable the **Trulog On Error** option to fully benefit from this feature.

Customizing Error Messages in HTML

Lists of specific error messages are contained in the `AppErrors.csv` and `AlertSeverities.csv` files, which can be edited by double-clicking them in the **Data Files** section of the project tree view. These lists are based on significant consulting experience and meet the needs of most users. If however you feel that certain error messages are missing, or you wish to add additional error messages, you can customize these files.

Customizing Error Messages in HTML

The `AppErrors.csv` file contains error messages that cause errors to be raised when they occur in HTTP responses. Each row defines one error message across three columns:

- Severity: Specifies the severity of error that is to be raised. Values in this column must begin with S, I, W, E, or T (signifying the severities `Success`, `Informational`, `Warning`, `Error`, or `Transaction Exit`).

- Data or Html: Specifies whether the entire HTTP response should be searched for the error message (API function `WebVerifyData`), or whether only the visible HTML content should be searched for the error message (API function `WebVerifyHtml`). Values in this column must begin with either D or H.
- Error String: The text of the error message. If you want to catch a set of error messages that have a common substring, it is good practice to enter only one entry where the `Error String` is the common substring. Example: If you have 100 different error messages, which all begin with `Microsoft SQL error:`, it is sufficient to have one list entry with the `Error String` `"Microsoft SQL error:`.

The `AppErrors.csv` file can be edited to meet your needs.

Customizing Alerts

Alerts are pop-up windows that are implemented by a JavaScript function called `Alert`. This function is called from the onload section of HTML pages.

Example: Alert in an HTML response

```
<body·class='PSPAGE'··onload="
  processing_main(0,3000);
  setKeyEventHandler_main();
  self.scroll(0,0);
  setEventHandlers_main('ICFirstAnchor_main',
                      ·'ICLastAnchor_main',·false);
  setupTimeout();
alert(&#039;Highlighted·fields·are·required.·(15,30)
>
```

The error detection mechanism detects any alert contained in a HTML response and treats it as being an error of `SEVERITY_TRANS_EXIT` severity. Known alerts that are to be ignored or reported with another severity can be specified in the `AlertSeverities.csv` file.

Each known alert message is represented by a single row in the `AlertSeverities.csv` file, across two columns:

- Severity: Specifies the severity of error that is to be raised. Values in this column must begin with N, S, I, W, E, or T (signifying the severities `None (=ignore)`, `Success`, `Informational`, `Warning`, `Error` or `Transaction Exit`).
- Alert String: The text of the alert string (or fragment thereof).

Alert messages are reported with the actual alert text, prefixed with the `gsAlertMsgPrefix` global variable. This variable contains an empty string by default, but can be assigned any value at anytime during script execution to meet your reporting needs.

Parameterization

The recorder can create variables for certain strings that appear in scripts. This makes script customization and randomization easier.

Sign-In Data

The user names and passwords that are used for sign-in are created as variables at the top of a script. The recorder also records a commented call to the `GetLoginInfoPS` function in the `TInit` transaction. The `GetLoginInfoPS` function retrieves values for the `gsUserId` and `gsPassword` variables from the `LoginPS.csv` file, which can be edited by double-clicking it in the **Data Files** section of the project tree view.

Customization of login data is done by uncommenting the recorded `GetLoginInfoPS` function call and populating the `LoginPS.csv` file with valid user accounts from your PeopleSoft application.

Example: Sign-in data created as variables

```
var
  gsUserId : string init "Admin";
  gsPassword : string init "Secret";
  // ...

  transaction TInit
    begin
    // ...
    PeopleSoftInit();
    //GetLoginInfoPS("LoginPS.csv", gsUserId, gsPassword);
    // ...
  end TInit;

  transaction TMain
  begin

  // ...

    SignInPS("login", LOGIN001, "EMPLOYEE"); // Form 1

  // ...

  end TMain;

dclform

  // ...

  LOGIN001:
    "httpPort"       := "" <USE_HTML_VAL> ,
                     // hidden, unchanged, value: ""
    "timezoneOffset" := "-120",
                     // hidden, changed(!)
    "userid"         := gsUserId,
                     // changed, value: "Admin"
    "pwd"            := gsPassword,
                     // changed, value: "Secret"
    "Submit"         := "" <USE_HTML_VAL> ;
                     // unchanged, value: "Sign In"
```

Input Values

Form field values that begin and end with underscores, or begin with `inp.`, are treated as input values. The recorder generates variables at the top of a script for these values. this makes randomization of input values easier.

Example: Variables created for input values

```
var
  gsInput_NewProduct : string init "_New Product_";
  gsInput_inp_Customer : string init "inp.Customer";

  // ...

dclform

  // ...

  MAIN009:
    "ICType"          := "" <USE_HTML_VAL> ,
                  // hidden, unchanged, value: "Panel"
    "ICElementNum"    := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: "0"
    "ICStateNum"      := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: "6"
    "ICAction"        := "#ICSave",
                      // hidden, changed(!)
    "ICXPos"          := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: "0"
    "ICYPos"          := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: "0"
    "ICFocus"         := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: ""
    "ICChanged"       := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: "0"
    "ICFind"          := "" <USE_HTML_VAL> ,
                      // hidden, unchanged, value: ""
    "PROD_ITEM_DESCR" := gsInput_NewProduct,
                      // changed, value: "_New Product_"
    "PROD_ITEM_EFF_STATUS" := "A",
                          // added
    "RBF_ARRA_WRK_PROD_ATTR_CATEGORY" := "",
                                      // added
    "PRD_KIT_ARR_FLG1" := "" <USE_HTML_VAL> ,
                       // unchanged, value: "S"
    "DESCR$0"         := "" <USE_HTML_VAL> ,
                      // unchanged, value: ""
    "RBF_ARRA_PRD_VW_DESCR$0" := "" <USE_HTML_VAL> ,
                              // unchanged, value: ""
    "RBF_PROD_ARGMT_RAISE_PCT$0" := "" <USE_HTML_VAL> ,
                                 // unchanged, value: ""
    "RBF_PROD_ARGMT_REDUCTION_PCT$0" := "" <USE_HTML_VAL> ,
                                     // unchanged, value: ""
```

HLS Support

HTTP Live Streaming (HLS) is an HTTP-based communications protocol for media streams. HLS has been widely adopted for video and audio delivery across the industry around the world. No matter if embedded in a web page or consumed through a standalone video player, Silk Performer automatically detects HLS traffic during recording and simulates a video player consuming HLS data during playback.

Silk Performer provides the BDL functions `HlsInit` and `HlsPlay` to load test audio and video delivery systems in on-demand or live streaming scenarios.

Silk Performer provides comprehensive statistics and metrics for streams and reliably detects stoppages due to bandwidth constraints or server issues.

Load Testing in Specific Environments

Silk Performer supports testing within a variety of computing environments.

Silk Performer CloudBurst

Overview

Silk Performer CloudBurst offers an affordable and flexible approach to confirming whether the system under test meets your performance requirements. CloudBurst enables you to rent virtual infrastructure, virtual users, or a combination of the two for as long as you need them.

With CloudBurst you can run load tests without permanent licenses. You are charged on a per-use basis, up to a number of concurrently executed virtual users within a 24-hour testing period.

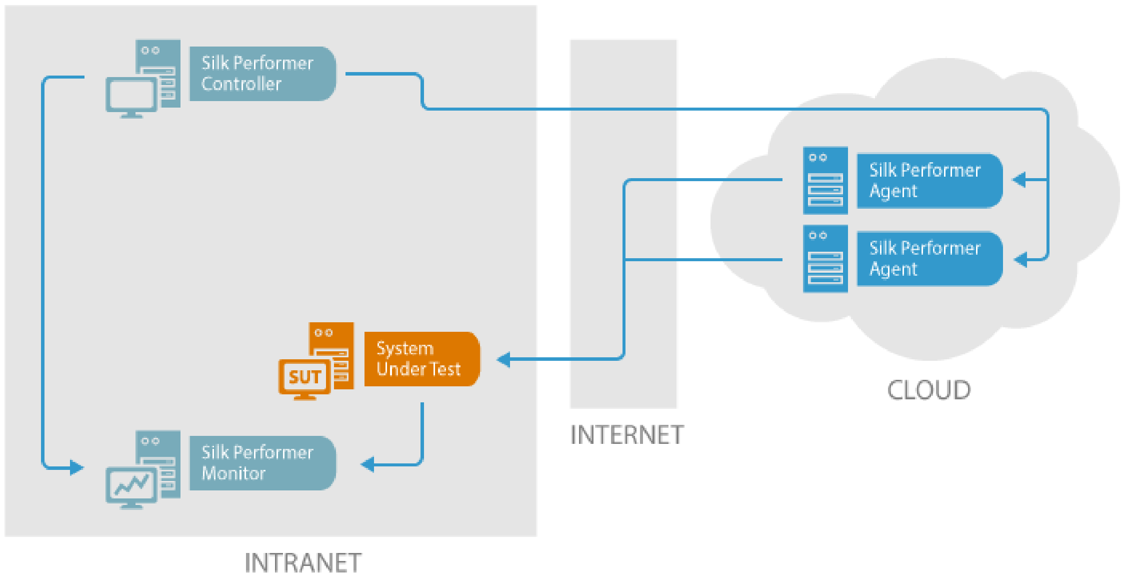
The CloudBurst virtual infrastructure is made available to you in the form of pre-configured, ready-to-use cloud agents, which can be deployed across multiple geographical regions. You can even combine pay-per-usage CloudBurst virtual agents with other agents you may have access to in your on-premise testing infrastructure (using a permanent Silk Performer license), all within the same load test.

If you choose to run tests with workload that requires additional virtual users beyond what your permanent Silk Performer license offers, you can use your CloudBurst license to deliver the additional virtual users on a pay-per-use basis.

CloudBurst Infrastructure

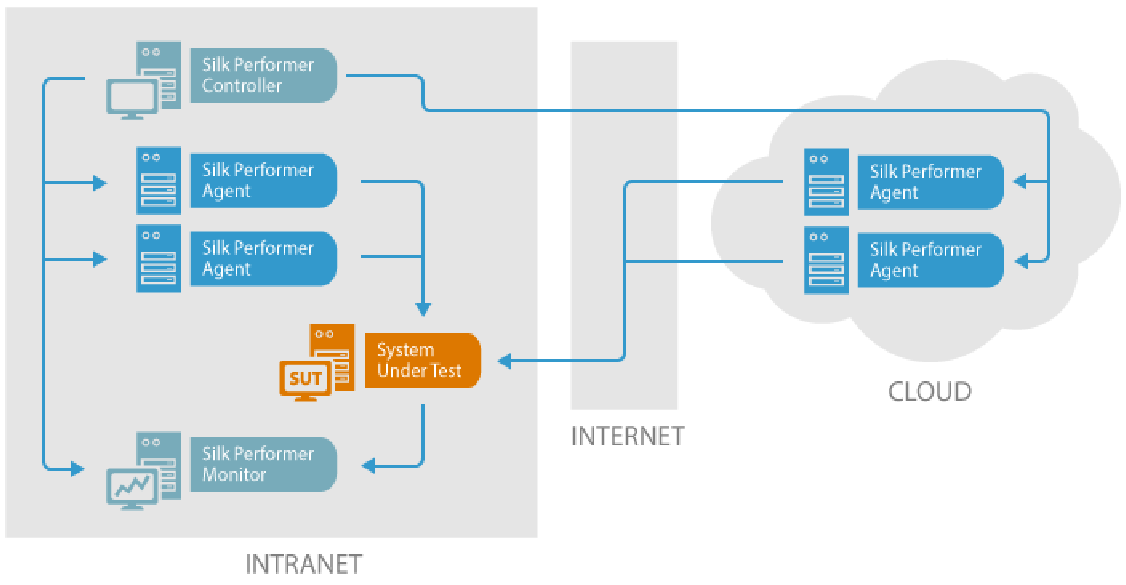
Cloud computing and Silk Performer CloudBurst extend the capabilities of traditional load testing. Silk Performer CloudBurst allows you to deploy agents in the cloud and let them load test all your applications, no matter if they are hosted on-premise or in the cloud. In addition, you can use a CloudBurst VPN to test your internal (non-Internet-facing) applications.

The diagrams below illustrate the basic concept behind CloudBurst. The first diagram shows how the cloud is used as a platform for Silk Performer agents that run tests against an Internet-facing application (the System Under Test).



The second diagram shows a hybrid scenario: Besides the agents that reside in the cloud, additional local agents, which reside in the local intranet, are used. This allows you to set up a number of agents that access your System Under Test from various locations. With the help of CloudBurst, you can define a broad range of regions, which provides you with comprehensive performance data.

Furthermore, such a combined setup provides you with the best possible test results: While cloud agents provide a realistic simulation of end-users through the Internet (and also help to detect performance issues), the local agents can be used for conventional performance testing and for identifying performance issues and bottlenecks.



Advantages of Using CloudBurst

Load testing with cloud agents offers a number of advantages:

- **Availability**: Cloud agents are available within minutes. You save time, because no setup or maintenance work is required, as it is with physical machines.
- **Scalability**: The number of cloud agents is scalable. No matter if you need hundreds or just two of them - you can increase or decrease their number at any time. By default, Silk Performer offers a certain amount of cloud agents. If you require more cloud agents, get in touch with support.
- **Deployment**: Cloud agents can be deployed easily and quickly. You can choose your cloud agents from a variety of regions across the globe.
- **Maintenance Cost**: With CloudBurst, you save maintenance cost: You only pay for the time your cloud agents are actually running.
- **Cloud Agent Manager**: All cloud-related settings can be managed easily with a single intuitive tool: the Cloud Agent Manager.

CloudBurst Offerings

Silk Performer CloudBurst is available in two forms:

Silk Performer CloudBurst SaaS is an online, self-service offering that does not require you to speak with a sales representative. To take advantage of this service, register on the *Micro Focus Build Portal*. You can purchase Micro Focus Credits using your credit card. Once you have filled your pre-paid account with Micro Focus Credits, you can use the credits to purchase the virtual infrastructure and/or virtual users you need to successfully run your performance tests.

Before you launch virtual agents or consume virtual user resources from CloudBurst for your load tests, Silk Performer will confirm your approval that your CloudBurst account can be debited.

Terms and conditions of CloudBurst SaaS are available on the *Micro Focus Build Portal*.

Silk Performer CloudBurst Enterprise is designed for enterprise customers and requires that you contact a sales representative. CloudBurst Enterprise offers the same functionality as CloudBurst SaaS, however different terms and conditions apply. Silk Performer CloudBurst Enterprise is available as prepaid model similar to the self-service model in CloudBurst SaaS and as pay-as-you-go model, where you are charged monthly according to your Micro Focus Credits consumption.

Working with CloudBurst

Cloud Agent Manager

To start the Cloud Agent Manager, click **Tools** in the Silk Performer menu and click **Cloud Agent Manager**. Then, login to an existing account or register.

The Cloud Agent Manager is an easy to use tool for managing all your CloudBurst tasks. It provides the following features:

- **Reserve Addresses**: Reserve IP addresses for your cloud agents. This ensures that your cloud agents keep the same IP addresses, which prevents you from regularly recurring work like specifying IP address exceptions for your firewall.

  Note: IP addresses are perpetually reserved for 24-hour time periods until you release the IP addresses.
- **Release Addresses**: Release IP addresses you have had reserved.
- **Start Agents**: Start cloud agents to use them in a load test. On the **Start Agents** dialog, you can specify the number of agents per region as well as how and when the agents are supposed to be shut down.
- **Stop Agents**: Manually stop agents when you no longer need them. On the **Manage your Cloud Agents** dialog, you can also see when the agents are automatically shut down, if an automatic shutdown behavior had been specified.
- **Configure VPN**: Configure a CloudBurst VPN to enable testing a machine that is part of your secured company network.
- **Refresh**: Refreshing the displayed data is useful as it may take some time before agents you start become available in the cloud.
- **Logout**: You can find the logout button on the bottom right of the window.

Close the window to exit the **Cloud Agent Manager**. The list of available agents on the **Workload Configuration** dialog box is updated automatically.

CloudBurst Connectivity Requirements

Controller

To use CloudBurst, a controller machine must be connected to the Internet by one of the following methods:

- direct connection
- HTTP Proxy (no authentication)
- HTTP Proxy (basic authentication)
- HTTP Proxy (NTLM authentication)
- HTTP Proxy (Kerberos authentication)

If a proxy server is present, the server must allow the HTTP CONNECT method to port 443.

Cloud Agents

A cloud agent is a virtual image. Communication between the controller and cloud agents is conducted over an HTTPS connection. The cloud agent allows for unrestricted outbound traffic. Inbound traffic is limited to port 443 and the HTTPS protocol.

Downloading Results from Cloud Agents

When a load test is completed, Silk Performer automatically downloads all results from your cloud agents. However, when the connection to the cloud agents is lost, the following applies:

- If the connection is lost during the execution of a load test, the controller tries to retrieve the results when the load test is completed. However, these results are not merged with the rest of the results. In such a case, Silk Performer will notify you with a message that results from disconnected agents are available in the results folder.
- If the connection is lost during the download process, Silk Performer will try to restart the download process and will add the results to the rest of the results. Usually, as a user, you will not notice any difference.
- You can also manually download results: In the **Cloud Agent Manager**, right-click an agent and click **Retrieve Results**. A dialog opens, containing a list of all the results. Specify a target folder and download the results to your machine.

Configuring Java for Cloud Agents

If you want to use Java-related technology on cloud agents, you might have to define the path to the java runtime environment (JRE). The 64-bit JRE is located in `C:\Program Files\Java\JRE`. The 32-bit JRE will be found automatically.

CloudBurst VPN

Company networks are usually protected by one or more firewalls, which typically block any unknown incoming traffic. If you intend to use cloud agents to test an application on a server environment that is part of your company network, you have the following options:

- Make the system under test accessible from the internet by configuring the company firewall for port forwarding or DMZ operation. Your IT department will help you with this task. Note that due to restrictive company policies, this option might be difficult to implement.
- Configure a CloudBurst VPN, a virtual private network between the cloud agents and a routing device within the company network, which allows the agents to access the system under test in the company network via proven and secure VPN technologies. The routing device forwards the incoming traffic between the system under test and the cloud agents.

To learn how to correctly configure a CloudBurst VPN, see *Configuring a CloudBurst VPN*.

Note: CloudBurst VPN is based on the UDP (User Datagram Protocol). It works through firewalls, which allow to establish UDP communication initiated from the router appliance (on-premise). Furthermore, operation through an HTTP proxy is not supported.

Basic Concept Behind CloudBurst VPN

Note: This topic is supposed to help you understanding the basic concept behind CloudBurst VPN. To learn how to actually configure a CloudBurst VPN, see *Configuring a CloudBurst VPN*.

The basic steps to configure a CloudBurst VPN are:

1. Open the CAM (Cloud Agent Manager)
2. Add a VPN configuration
3. Download the VPN router appliance and install it on the routing device
4. Start one or more cloud agents. Once the VPN is operational running cloud agents will automatically join it. Note that you can start agents any time, before or after the VPN configuration

The following paragraph explains what happens in the background when a VPN is configured. Take a look at the graphic below to get a better idea of the concept behind the process:

1. When you open the CAM and start cloud agents, the CAM connects to the Silk Performer CloudBurst Service and requests cloud agents to be launched.
2. The CloudBurst Service updates the CAM about all cloud agents and their status.
3. When you add a VPN configuration, the CAM connects to the CloudBurst Service, which tells the Cloud Agents to act as VPN Servers.
4. When you download the VPN router appliance and install it on the routing device (Router), the device acts as VPN Client.

5. The VPN Client/Router requests the VPN details from the CloudBurst Service along with the list of available cloud agents.
6. The VPN Client/Router connects to the Cloud Agents/VPN Servers.
7. The Cloud Agents/VPN Servers connect to the specified DNS Server to check if the company network can be reached.
8. When you start a load test, the Cloud Agents can use the connection through the Firewall and through the VPN Client/Router to put load onto the System Under Test (SUT).

Configuring a CloudBurst VPN

1. In the main menu, click **Tools** > **Cloud Agent Manager**.
2. Login to your CloudBurst account with your user name and password.
3. Click **Start Agents**, enter the desired amount of agents per region, and click **Start**. The **Status** column displays **Starting**. When the agents are started, the column displays **Ready**.
4. Click **Configure VPN** and click **Add**.
5. Enter a **Name** for your VPN configuration.
6. Enter the hostname of your **System Under Test** (SUT) and click **Get network settings**. The Cloud Agent Manager (CAM) then tries to resolve the IP address of the SUT. Based on that IP address, all company network settings are filled in automatically.
7. If the SUT cannot be reached, an information dialog box displays. This is generally the case when your machine and the SUT are not in the same subnet and when your machine is not allowed to access the subnet of the SUT. In such a case, fill in the company settings manually.

   Note: If you have problems finding out the correct settings, contact your system or network administrator.
8. Make sure that **Activate VPN immediately** is checked and click **OK**. The **State** column displays **Preparing VPN**.
9. When the **State** column displays **Waiting for router**, click the link **download router image**. This will download an appliance for the virtual machine software VirtualBox.
10. Transfer the appliance to the routing device.

    Note: The router appliance must be located in the same network segment as the System Under Test (SUT). If this is not possible, contact TechSupport for further instructions.
11. If you have not installed VirtualBox on the routing device, download the setup file from the VirtualBox website (*https://www.virtualbox.org/*) and install it.
12. Start VirtualBox, import the appliance and start it.

    Note: Make sure to assign new MAC addresses to all network cards before you import the appliance.
13. Enter your CloudBurst credentials. A Linux operating system starts in Virtual Box. This makes the routing device a VPN client.

Once the Linux operating system runs, the Cloud Agent Manager (CAM) displays that the connection to the router is established. Your CloudBurst VPN is set up.

VPN Router Appliance

During the process of configuring a CloudBurst VPN, you have to download the VPN router appliance and transfer it to the routing device. The VPN router appliance is a Linux appliance for the virtualization software VirtualBox. Hence, make sure that VirtualBox is installed on the router beforehand.

Then, import the appliance in VirtualBox and start it. VirtualBox will start a Linux operating system. Once Linux is running, the routing device acts as VPN client. The VPN client is then part of the CloudBurst VPN and routes all incoming traffic from the cloud agents to the specified subnet within the company network.

Note: The router appliance must be located in the same network segment as the System Under Test (SUT). If this is not possible, contact TechSupport for further instructions.

Micro Focus Credits

Micro Focus Credits are a virtual currency that can be used to purchase CloudBurst testing services, available from the *Micro Focus Build Portal*. The billing process follows a pre-paid model: You first purchase Micro Focus Credits and then decide how and when you want to spend them.

The price per Micro Focus Credit depends on how many credits you buy at one time. There are several credit packages to choose from.

Note that Silk Performer CloudBurst provides an invitation system: On the *Micro Focus Build Portal*, you can invite other CloudBurst users to join your CloudBurst account. This allows the other CloudBurst users to make use of the Micro Focus Credits that are purchased for your account.

CloudBurst Load Test Fees

The amount of Micro Focus Credits required to purchase a CloudBurst test is comprised of infrastructure rental fees and virtual user license fees.

Infrastructure Rental Fees

There are three types of infrastructure rental fees:

- active infrastructure rental fee
- passive infrastructure rental fee
- fee for IP address reservation

Infrastructure rental fees are paid for each hour following the start of a cloud agent. Each started hour is charged as a full hour. For example: If an agent runs for 50 minutes, you are charged for 1 hour. If the agent runs for 1 hour and 5 minutes, you are charged for 2 hours.

The infrastructure rental fee is comprised of a passive and an active fee. The passive fee is charged for the time an agent runs, the active fee is charged for the time the running agent is used to execute a load test. During the execution of a load test, both the passive and the active fee are charged. For example: You start a load test with a simulation time of 2 hours. When the load test is complete, you prepare for a second load test, which takes you 1 hour. In the meantime, you leave the agent running. Then you start the second load test with a simulation time of 30 minutes. For this scenario, you are charged a 4-hour passive fee and a 3-hour active fee. Note: The 30 minutes are calculated as a full hour.

Note: If you run out of Micro Focus Credits while using the CloudBurst infrastructure, you will be notified by email 24 hours before your agents are shut down automatically.

Reserving IP addresses for your cloud agents also causes infrastructure rental fees. The current rate for reservation displays in the **Cloud Agent Manager**. Reserving IP addresses prevents you from regularly recurring work like specifying IP address exceptions for your firewall.

Note: IP addresses are perpetually reserved for 24-hour time periods until you release the IP addresses.

Virtual User License Fees

Virtual user license fees are calculated based on the maximum number of concurrently active virtual users consumed within a 24-hour testing period. The fee per active virtual user is largely dependent on the application type under test. For example: If you start a load test with 100 virtual users, you are charged a 100 virtual users license fee for the 24-hour testing period. When the load test is complete, you start another load test with 150 virtual users. Now you are charged a fee for the 50 additional virtual users. For the next load test, you configure to use 120 virtual users. This test will not cause an additional fee, since the already debited fees cover up to 150 virtual users. This is true if you execute all three load tests from this example within the 24-hour testing period.

For more information about current rates, visit the Micro Focus website.

Spanning Testing Periods

The 24-hour testing period model allows you to run a load test, analyze the results, and rerun the load test multiple times with adjusted settings. Within the 24-hour testing period, you can optimize the settings of your load test and rerun the test without paying for each additional test run.

If a test, which is started within one 24-hour testing period, would extend into the next testing period, a new 24-hour testing period is automatically created with the start of the load test.

Note: The maximum simulation time for CloudBurst load tests is 24 hours.

Estimating and Debiting Micro Focus Credits

Estimating Micro Focus Credits Consumption

Before you start a load test, Silk Performer displays an estimate of how many Micro Focus Credits you will likely be charged by the end of the load test. The estimate takes all virtual user license fees that are covered in the current testing period into account. However, it does not include any unforeseen charges that may be related to changes in simulation time or concurrently executed load tests.

If you use the dynamic workload model, which requires you to stop tests manually, the infrastructure rental fee estimate will be equal to the required Micro Focus Credits per started hour.

Debiting Micro Focus Credits

When a test is complete, Silk Performer calculates the amount of Micro Focus Credits that is to be debited from your account. This calculation takes into account the actual simulation time, which is used to calculate the active infrastructure rental fee, and the testing period boundary for virtual user license fees.

A detailed balance sheet is available from the Silk Performer **Load Test Summary** page, which is displayed following each test.

Developing Performance Tests in Visual Studio

The Silk Performer Visual Studio extension allows you to develop performance tests in Microsoft Visual Studio. It offers functionality to record web technologies, supporting both the protocol-level as well as the browser-driven approach. The recording functionality of the Visual Studio extension resembles the one of the Silk Performer Workbench and includes capturing web traffic and user interactions. The captured information can be converted into a C# class including methods, each representing a user transaction.

The Visual Studio extension offers a comprehensive C# binding for all kernel, web, and browser API functions. Thus, it combines the rich development feature set of Visual Studio, including all debugging features, with the powerful load testing capabilities of Silk Performer, including capture file recording and TrueLog analysis.

Tests created with the Visual Studio extension can be executed either within Visual Studio for trial runs, or within Silk Performer for concurrency and load testing scenarios. When exporting C# projects to Silk Performer, a short BDL script stub is generated, containing calls into the compiled C# code (assembly).

To learn how to install the Visual Studio extension, refer to the Installation Guide.

The C# Binding

Currently, Silk Performer supports all kernel, web, and browser BDL functions for C# binding. In Visual Studio you can access these functions by calling their equivalent static C# class methods, such as `Kernel.MeasureStart()`, `Web.PageUrl()`, or `Browser.Click()`.

In other words: For each built-in Silk Performer .bdh file, you will find a corresponding C# class in the PerfRunDotNet assembly. And for each API function defined in that .bdh file, there is an equivalent static method in the corresponding C# class.

C# script structure

A C# class that is intended to be used for load testing in Silk Performer contains a `VirtualUser` attribute and a `CodePage` attribute. `VirtualUser` specifies a `UserName` and a `ScriptName` property. `CodePage` specifies the codepage of the recording system. Methods with the `Transaction` attribute will be mapped to transactions when exported to Silk Performer.

```
namespace SilkPerformerRecorder
{
  [VirtualUser(UserName = "VUser", ScriptName = "MyScript")]
  [CodePage(1252)]
  public class MyScript
  {
    [Transaction(ETransactionType.TRANSTYPE_INIT)]
    public void TInit()
    {
    }

    [Transaction(ETransactionType.TRANSTYPE_MAIN)]
    public void TMain()
```

Data Type Mapping

To pass parameters from .Net/C# to BDL and vice versa, some built-in data types have to be converted.

Strings

While Silk Performer and its BDL are based on the multi-byte character set (MBCS) encoding, the C# language works with Unicode. Thus, every string has to be converted when passed between the two worlds. The PerfRunDotNet assembly works with the BdlString class, which handles all these conversions. Moreover, it also takes care about binary buffers. For convenience, the BdlString class is assignment compatible with the C# string.

Sizespec and lenspec variables have been removed in the C# mapping, since C# strings typically know their length.

Static parameter values

Some static integer values, like severity parameters, have been replaced by specific enums.

Forms

BDL forms are mapped to a `BdlForm` class in C#:

```
private static readonly BdlForm ZIP_SEARCH001 = new BdlForm()
    {
      Entries = new BdlFormEntries()
      {
        { "zip-search", "", Kernel.USE_HTML_VAL }, // hidden, unchanged, 
value: "zip-search"
        { "zip-search:zipcode", "", Kernel.USE_HTML_VAL }, // unchanged, 
value: ""
        { "javax.faces.ViewState", "", Kernel.USE_HTML_VAL }, // hidden, 
unchanged, value: "j_id3:j_id4"
        { "zip-search:search-zipcode.x", "62" }, // added
        { "zip-search:search-zipcode.y", "4" } // added
      }
    };
```

Code page dependent replay

Due to Silk Performer being a MBCS-based application, the recorder automatically generates a codepage attribute in the C# script, which gets carried over to the .bdf script stub. It also takes care about converting all Unicode strings that cannot be represented in the system codepage into hex-byte arrays.

If the `CodePage` attribute is defined in a C# script, the following applies:

- All classes in the project must have the same codepage.
- Each string used in the C# script has to be convertible to MBCS strings using that codepage.
- The defined codepage is used to convert C# strings (Unicode) into MBCS strings, and also to interpret the hex byte arrays generated by the recorder. Therefore the codepage attribute should always be set, especially if the recorder generated hex byte arrays when encountering otherwise unprintable characters.
- A compiletime-check makes sure that all strings work for the defined codepage.

If the `CodePage` attribute is not defined in a C# script, the system codepage is used during replay.

Licensing

The licensing of load tests with C# test scripts is similar to the licensing model that applies when using BDL as scripting language. A single user run is always license-free. During a trial run, no license is being checked out. When executing a load test using C# scripts, Silk Performer checks for the technologies used in the scripts and maps them to equivalent feature license type web, standard, or premium. The mapping is the same as with BDL.

If however a C# script is used for .Net testing rather than driving one of the supported Silk Performer technologies, a standard license will be checked out.

Capture File Compatibility

Capture files created during a recording session in Visual Studio are fully compatible to capture files created directly within the Silk Performer Workbench. Thus, you can add a capture file to a C# project - by dragging it onto the Visual Studio project tree - and generate a C# script.

You can also reuse a capture file created in Visual Studio by adding it to a Silk Performer project manually and then generating a BDL script out of it.

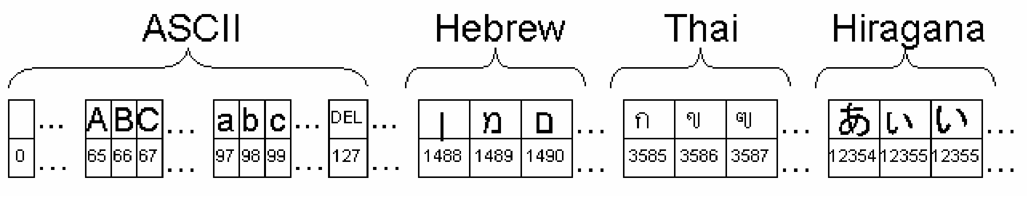
Advanced Topics

Note the following when working with the Visual Studio extension.

- If the C# code is compiled in debug mode and debug symbols are available, those are used to add information about the current position in the C# script also to nodes in the replay TrueLog. In this case, the current position in the C# script is shown in the **Script** tab of TrueLog Explorer instead of the current position in the .bdf stub script.
- In Visual Studio you can create a C# or VB.Net project and code manually. Note however, that the recorder only prints C# code. The recording feature is only available if a C# project is used.

Multibyte Support

Unicode is a list of all known characters. It includes all alphabets of all spoken and unspoken languages. Each character has its own unique index in the Unicode list. The first 128 characters are known as ASCII characters.



When data is stored or computed, the Unicode list is not used to represent the characters. Instead, so-called *character encodings* (or character sets) define how characters are represented on computers and within files. Numerous character sets are used throughout the world.

Two of the more frequently-used character encodings that cover all Unicode characters are:

- *UTF-8*: Requires 1-4 Bytes per character; this character set is widely used for international Web sites and international text representation. Strings in the Linux kernel and Java are encoded in UTF-8.
- *UTF-16*: Requires 2 or 4 Bytes per character; this encoding is mainly used for string representation in the Windows NT kernel (Win NT and newer).

Most character sets do not define a representation for all characters in the Unicode list. Rather, they define a subset of characters that are used in a specific regional area of the world. These are commonly referred to as *code pages* (cp):

- *ASCII*: Single Byte encoding: 1 Byte per character
- *Latin-1 (Windows Codepage 1252)*: Single Byte encoding: 1 Byte per character
- *Shift-JIS (Windows Codepage 932)*: Double Byte encoding: 1 or 2 Bytes per character
- *EUC-JP (Windows Codepage)*: Includes 3 Japanese char sets: 1, 2, or 3 Bytes per character

When developing applications for Windows, programmers can choose between Unicode (UTF-16) string representation or *Multi-Byte-Character-Set (MBCS)* string representation. MBCS representation refers to a geographic region-dependent code page encoding (for example, *Shift-JIS* for Japan; *Latin-1* for the Americas and most European countries). This affects all GUI elements, as all data needs to be displayed in the same string representation that has been selected for the application.

Multibyte Support in Silk Performer

Silk Performer is a MBCS-based application, meaning that to be displayed correctly, every string must be encoded in MBCS format. Because Silk Performer, and particularly TrueLog Explorer, visualize and customize data that originates from Web servers with different encodings (UTF-8, EUC-JP, ISO-8859-1, and so on), many string conversion operations may be involved before data can be displayed.

Sometimes when testing UTF-8 encoded Web sites, data containing characters cannot be converted to the active Windows system codepage. In such cases, Silk Performer’s Web Recorder scripts all non-convertible strings into hex format. If the data is only displayed, TrueLog Explorer will replace the non-convertible characters with a configurable *replacement character* (usually '?').

Example

On an English Windows system with Japanese local settings (system codepage is Shift-JIS) a UTF-8 application is recorded. For some reason a link name on a particular Web page contains a Korean character. Silk Performer’s Web Recorder attempts to convert the string from UTF-8 to Shift-JIS to make it readable in the script. The conversion fails because of the Korean character, which does not occur in the Shift-JIS code page. The Web Recorder then uses hex notation for the link name in the script.

Codepage Check

One of the most important things to know about Silk Performer's multibyte support is that the system codepage of the recording machine must match the system codepage of the replay machine. This is necessary for the replay engine to correctly interpret the byte sequences in the script as characters and strings.

Therefore the Silk Performer Recorder scripts the `@codepage` annotation into the script along with the currently active system codepage. Whenever this annotation is present in a script the Silk Performer runtime system checks if the system codepage of the machine matches the one specified in the script. If they do not match the runtime stops and an error message displays. However, you can add a second parameter to the `@codepage` annotation with the value `false`. In this case an information displays but the execution continues.

Syntax

```
@codepage( in codepage        : number,
           in abortOnMismatch : boolean optional := true )
```

Example

```
@codepage(123)          // aborts the execution when the codepages do not
match
```

```
@codepage(123, false)   // an information indicates the mismatch, but the
execution continues
```

API Functions That Require Additional Encoding

Regardless of the character encoding that a Web site uses, Silk Performer can only visualize multi-byte character set (MBCS) strings. However, several API functions require the encoding of some string parameters in the encoding that the server expects. Silk Performer's Web Recorder scripts `SetEncoding()`, `ToEncoding()`, and `FromEncoding()` functions at appropriate locations so that replay engines can know the required encoding. Typically, parameters that require an appropriate encoding, such as link names, if they refer to data that needs interpreting. Parameters that are not a part of client-server communication, such as timer names, do not require conversion.

The `SetEncoding()` function sets an encoding for all subsequent `ToEncoding()` and `FromEncoding()` functions. The encoding itself is passed as the name of the encoding as it appears in the charset specification of the Web page, such as `UTF-8`, `SHIFT_JIS`, `WINDOWS-1252`, `WINDOWS-1255`, or `EUC-JP`.

`ToEncoding()` converts the string that is passed as a parameter, which is encoded in the system codepage (MBCS), to the encoding specified by the last `SetEncoding()` function.

`FromEncoding()` converts the string that is passed as a parameter, which is encoded in the encoding specified by the last `SetEncoding()` call, to the system codepage (MBCS).

Copying Strings into Silk Performer

When copying strings into Silk Performer, the paste operation checks if the data:

- Needs to be converted to the MBCS (system codepage)
- Can be converted to the MBCS
- Cannot be converted and so needs to be opened with Silk Performer’s Unicode Text/Hex Editor

If copied data is, for example, UTF-8 encoded and the paste operation fails to convert it to the system codepage, Silk Performer launches its Unicode Text/Hex Editor. The tool is a Unicode application and is therefore able to display any string, regardless if the characters exist in the system codepage. The editor can also display the hex notation of strings in order to paste them into a BDL script.

Network Emulation

Network Emulation Settings

Whenever a client and a server need to communicate, a connection between the two is established through one or more networks. This network connection is used to send and receive data, which typically is split into data packets.

The connection between client and server can comprise various networks with vastly diverse characteristics. This can result in a delayed transfer of data packets (latency), in a loss of data packets (packet drop rate), or in a reduced transfer rate of one of the networks (bandwidth).

Silk Performer allows you to emulate complex network connections by configuring a range of settings and parameters:

- The **Latency** is the time that is required for a data packet to travel from one endpoint of the network connection to the other. It depends on various factors, such as the physical speed limit of the network medium (fiber, wire, over-the-air), the bandwidth restrictions and the distance between the two endpoints. The latency of a network connection can vary over time. This variation is called jitter. For typical web load testing scenarios jitter is of less importance as network emulation parameter. Emulating jitter is not supported by Silk Performer.
- The **Bandwidth** is defined as the data transfer rate, bit rate or throughput. It is measured in bits per second (bit/s or bps), kbps, Mbps, or Gbps.
- The **Packet drop** rate: Occasionally data packets can get lost on the way from the sender to the receiver. The main reason for dropped packets is an overloaded network or network device. The TCP/IP (Transmission Control Protocol/Internet Protocol) has a built-in detection of dropped packets and guarantees that sent packets are delivered even if some get lost and need to be retransmitted. A high packet drop rate involves high administration effort on the TCP/IP level and thus has a significant effect on the data throughput.

Silk Performer offers presets (preconfigured settings) for a variety of wired and wireless network technology standards. You can adjust all settings to your needs to simulate specific network conditions.

The Network Emulation Driver

Silk Performer can emulate bandwidth limitation out of the box. For latency and packet drop emulation a dedicated network emulation driver is required. By default, the driver is not installed with Silk Performer. To use the full network emulation functionality, make sure to install the driver during Silk Performer setup on your agents.

Important: Enable **Use Network Emulation Driver** in the profile settings only if the network emulation driver is installed on your agents. You can then adjust the **Latency** and **Packet Drop** settings to your needs. If the driver is not installed and you enable this option, an error will occur during load test execution.

Note the following specifics:

- The network emulation driver is not compatible with all network adapters. In such a case, machines or network interfaces might no longer be accessible and the network might slow down.
- The network emulation driver is currently not supported for Windows 10.
- Network emulation is not supported for CloudBurst.

Windows Vista and Windows 7

Silk Performer is fully compliant with Microsoft *User Account Control (UAC)* guidelines. It has been designed such that it does not require administrative privileges at any point in its workflow, beginning with project definition and continuing through to test analysis. Even activities such as recording and the launching of remote agent processes can be done with standard user privileges.

The **System Configuration Manager** however, due to its system administration functions, does require elevation into administrator mode in accordance with UAC guidelines.

UAC Overview

With Windows Vista and Windows 7, Microsoft encourages use of standard user accounts for daily work rather than administrator accounts. Several changes have been made to Silk Performer privilege management to accommodate this change, though these changes are undetectable to users because they do not affect Silk Performer workflow.

Microsoft encourages users to follow their privilege-management paradigm with UAC. UAC informs users when an application leaves the standard user account privileges path and requires higher-level privileges, even when the user account already has the required privileges. For this reason, even administrators are prompted when performing tasks that require administrator privileges. Another consequence of UAC is that software developers are encouraged to write Windows applications that require as few privileges as possible for normal operation.

UAC on Agent Machines

Silk Performer agent processes can be run without administrative privileges. There are exceptions to this however (for example, if an executed script contains an action that requires administrative privileges). In some situations it may be useful or even necessary to launch an agent process under a particular administrator user account. By default however, agent processes are launched under the built-in `SYSTEM` account.

On machines running a Windows operating system that supports UAC, an administrator account does not automatically hold administrator privileges. Administrator accounts gain new privileges by prompting the user for elevation, which the user may then grant explicitly. By default there is one exception to this behavior: the built-in `administrator` account does not require manual elevation.

To start an agent process on a UAC-enabled machine, you have the following options:

- Use the `SYSTEM` account (default)
- Use the built-in `administrator` account and meet one of the following requirements:
  - Turn off UAC
  - Disable security policy `User Account Control: Run all administrators in Admin Approval Mode`. This can be configured using the Windows Local Security Policy tool (`secpol.msc`).
- Use a non-built-in administrator account and meet one of the following requirements:
  - Executed script does not require administrator privileges
  - Executed script does not perform administrative actions

Note: UAC functionality is relevant to the **Advanced** tab at **System Settings** > **Agents**. It is also relevant to **SysConfManager** > **Applications** > **User Credentials**.

Silk Performer Plug-Ins

Silk Performer offers a plug-in interface which extends Silk Performer’s compatibility to third-party technologies, enabling coordination and control of third-party technologies from within the Silk Performer environment.

The controls that are available for plug-ins vary based on the functionality offered by each plug-in. By default, each plug-in offers the following attributes through the Silk Performer **System Settings** dialog:

On the **General** page:

- plug-in name, version, and description details

- enabling/disabling of the plug-in
- a list of features supported by the plug-in

On the **Attributes** page:

- name, description, datatype, and value for each available attribute

Depending on available plug-in functionality, other tabs and configurable attributes may also be available.

Silk Performer offers a plug-in framework for third-party diagnostics tools. The framework enables third-party tool vendors to integrate into one or more of the following tools:

- Silk Performer

  The Silk Performer plug-in can be used to modify profile settings so that the runtime sends an additional HTTP header tag with each page request. The same functionality is available through the `WebSetHttpTag` BDL function.
- TrueLog Explorer

  When a user selects an API node, TrueLog Explorer passes the plug-in header and body, including the additional HTTP tag. TrueLog Explorer's lower-right window includes an additional tab for each enabled plug-in. The plug-in tabs are Internet Explorer controls, so plug-ins can present anything that can be expressed through HTML.

  Plug-ins can use the HTTP tag to identify traces and present complete component break-downs for each web page call.
- Performance Explorer

  Plug-ins that implement the PerfExp plug-in interface can be called from the (OVR) Overview Report/ Ranking section. Overview reports show plug-in related icons for each ranking measurement for which the plug-in can provide in-depth information.

Plug-In Initialization and Configuration

All available plug-ins are installed to the following directory during Silk Performer setup: `C:\Program Files\Silk\Silk Performer 20.0\Plugins`.

When Silk Performer is launched, it checks for installed plug-ins. All installed plug-ins are then loaded, verified, and initialized.

Plug-ins can be enabled, disabled, or configured by users via Silk Performer's **System Settings** dialog. The **System Settings** dialog adds a new icon in the left-hand pane for each available plug-in. On the **General** page, basic information (including description, version, and supported features) and the ability to enable/disable the plug-in is provided for each installed plug-in. Each plug-in is also provided an **Attributes** tab that lists all attributes associated with the plug-in. Some plug-ins may include additional tabs related to available functionality.

Plug-ins are disabled by default and must be enabled before use.

Message Output

A new tab is inserted into Silk Performer's **Output** window for each installed and enabled plug-in.

Silk Performer outputs all messages from plug-ins into these tabs. **Output** tabs are removed when plug-ins are disabled.

Output pages display the following information:

- Timestamp (HH:MM:SS).
- Severity ("Error," "Warning," or "Info").
- Numerical message code.
- The text of the message. Messages are displayed in the color that corresponds to their severity (Error = red; Warning = orange; Info = magenta).

AppDynamics Plug-In

The AppDynamics plug-In is installed along with Silk Performer setup. It is disabled by default.

AppDynamics focuses on monitoring performance at the Business Transaction level. Business transactions mirror end user activity and are defined as a category of user requests.

The current Silk Performer - AppDynamics integration supports automatic naming of such transactions based on Silk Performer timer names.

After successful configuration, an API call from Silk Performer such as the following:

```
BrowserNavigate("http://myserver.mydomain.com/login", "LoginPage");
```

Appears as the following business transaction in AppDynamics:

```
SilkPerformer.LoginPage
```

Configuring AppDynamics Plug-In Settings

1. In Silk Performer, select **Settings** > **System Settings**.
2. Select the **AppDynamics Plug-In** group icon.
3. On the **General** page, check the **Enable plug-in** check box.
4. Configure name/value pairs on the **Attributes** page as required. The default **Tag** value is `AppDynamicsTag`.

   Note:

   A number of integration settings are dependent on the Tag attribute retaining the value `AppDynamicsTag`. If you change this default value, you will need to change additional settings, as explained below.
5. Click **OK** to save your settings.

Configuring Automatic Transaction Naming in AppDynamics

1. In the leftmost AppDynamics navigation pane, click **Configure** > **Instrumentation**.
2. Select the **Use Custom Configuration for this tier** option button.
3. Scroll down to the custom rules section of the page.
4. Click **+** to add a custom match rule.
5. On the **Transaction Match Criteria** page, select **Servlet** from the drop-down list.
6. Specify the name of the custom rule. This name will become the first part of the generated transaction name.

   For **URI** select `Is Not Empty`.

   Select **Check for parameter existence** and type `AppDynamicsTag`, to match what as you configured in Silk Performer, into the **Parameter Name** field.
7. Select the **Split Transactions using Request Data** page and check the **Split Transactions using Request Data** check box.
8. In the **Apply a custom expression on HTTPServletRequest and use the result in Transaction names** field, add the following string:

   ```
   ${getHeader(AppDynamicsTag).substring(Int/3)}
   ```
9. Click **Save**.

Verifying the Integration

1. Within Silk Performer, start your test. After a few minutes, AppDynamics displays the transaction names in the **Business Transactions List**. When the test is complete, the Silk Performance Explorer Overview Report appears.
2. Compare the results of the test shown in the overview report alongside the AppDynamics' Business Transactions list.

How Timers are Matched with Business Transactions

HTTP requests executed by Silk Performer's replay engine are extended by an additional HTTP header. For example, the following API call:

```
BrowserFormSubmit("//FORM[@name='loginForm']", "Login");
```

The HTTP header `AppDynamicsTag` (the tag attribute you defined during Silk Performer/AppDynamics plug-in configuration) contains the specified timer name `NA=<TimerName>` (for this example, `NA=<Login>`.

AppDynamics automatic transaction naming parses the HTTP header by applying the above custom expression `“${getHeader(AppDynamicsTag).substring(Int/3)}”` to the `HTTPServletRequest` object and evaluates it to the following Java code:
`request.getHeader(“AppDynamicsTag”).substring(3)`

Note: The purpose of the substring method is to exclude the `NA=` portion of the HTTP header.

Requests containing the same `AppDynamicsTag` header are grouped into the same business transaction. For example, all requests with an `AppDynamicsTag: NA=Login` header would be grouped into a `Login` business transaction.

Advanced Configuration

The AppDynamics plug-in allows you to specify which tags are being set along with web requests in the additional HTTP header. For a detailed description of the tags refer tot the explanation in the settings UI.

The `Probability` attribute is a floating point value between `0` and `1`, specifying the percentage of virtual users that should send the additional HTTP header.

Dynatrace AppMon Plug-In

The Dynatrace AppMon plug-in is installed with Silk Performer and disabled by default. To work with this plugin you need Dynatrace AppMon Client to be installed on the controller machine. If enabled, the plug-in modifies profile settings so that the runtime sends an additional HTTP header with each page request.

This HTTP header contains load test specific data which is thus made available to Dynatrace AppMon. The Dynatrace AppMon plug-in also integrates with TrueLog Explorer: For the selected node in the TrueLog, the plug-in tab in the lower right window allows direct access to the corresponding PurePath in Dynatrace.

Note: The additional HTTP header is only added during protocol-level testing and browser-driven testing using Internet Explorer.

To configure the plugin:

1. In the Silk Performer menu, click **Settings** > **System**.
2. Click **Dynatrace AppMon**.
3. Check **Enable plug-in**.
4. Click the **Attributes** tab and configure **Name/Value** pairs as desired.

- The Dynatrace AppMon plugin requires the correct port to be configured to be able to communicate with the AppMon client. In addition, the system profile value must match the Dynatrace AppMon profile you want Silk Performer to work with.
- The tagging header name is expected to be `x-dynatrace`. Use `dynatrace` as header name only for very early versions of Dynatrace Diagnostics. For more details refer to the Dynatrace AppMon documentation.
- In the **Tag Selection** section you can choose the tags you want virtual users to send as part of the additional HTTP header with each request.

Dynatrace SaaS and Managed Plug-In

The Dynatrace SaaS/Managed plug-in is installed with Silk Performer and disabled by default. If enabled, the plug-in modifies profile settings so that the runtime sends an additional HTTP header with each page request. Besides, the plug-in can

- configure request attributes in Dynatrace to extract the load test specific information contained in this additional header, making it available for analysis in Dynatrace.
- inform Dynatrace about virtual user errors through the plug-in.
- add an annotation containing general load test information.
- configure request naming rules in Dynatrace based on the timer name parameters passed to Web* and Browser* API calls.

Note: The additional HTTP header is only added during protocol-level testing and browser-driven testing using Internet Explorer.

To configure the plugin:

1. In the Silk Performer menu, click **Settings** > **System**.
2. Click **Dynatrace SaaS/Managed**.
3. Check **Enable plug-in**.
4. Click the **Attributes** tab and configure **Name/Value** pairs as desired.
    a. Set the base URL to your Dynatrace environment.
    b. Generate an authentication token with the required permissions in Dynatrace and specify it in the plug-in attributes.
    c. Tag relevant components of your system under test in Dynatrace and specify the used tag as application tag in the plug-in attributes.

    Additionally, you can choose which configurations the plug-in should do for you and which tags to send in the additional HTTP header.

Silk Central Integration

Silk Performer is fully integrated with Silk Central's test-planning and test-execution functionality. Silk Performer projects can be integrated into Silk Central test plans and can be directly executed through Silk Central. This feature allows for powerful test-result analysis and reporting, and it provides for tests that are run automatically by Silk Central based on pre-configured schedules. This type of testing is called *unattended testing*.

Silk Central projects can be downloaded to Silk Performer, where scripts and settings can be edited. Edited projects can subsequently be uploaded to Silk Central to make them available for future test executions.

Note: The term *Project* is used differently in Silk Performer than it is in Silk Central. When uploaded to Silk Central, a Silk Performer project becomes the core element of a Silk Central *test definition*. Silk Central projects are higher-level entities that can include multiple test definitions, executions definitions, and requirements.

Silk Performance Explorer can be used for in-depth analysis of test runs. Performance Explorer can be launched directly from Silk Central's **Executions** unit by way of execution runs on the **Runs** page, either from Silk Performer or from Performance Explorer itself. For details regarding Performance Explorer's integration with Silk Central, refer to the Performance Explorer Help .

The results of Silk Performer test runs can be uploaded to Silk Central and associated with test definitions. To complete this task, Silk Performer searches its results directories and uploads the appropriate files to Silk Central.

For additional information about Silk Central's integration with Silk Performer and Silk Performance Explorer, refer to Silk Central Help and Silk Performance Explorer Help.

Configuring Silk Central Integration

1. In the Silk Performer menu, click **Settings** > **System** . The **System Settings** display.
2. Click the **Silk Central** tab.
3. In the **URL** field, specify the URL of the host on which Silk Central's front-end server is running. Make sure to specify the full URL including the port and the Silk Central instance.
4. Click **Use credentials** and type valid credentials for Silk Central in the fields **Username** and **Password**.
5. Alternatively, click **Use token** and type or paste the web-service token into the field. To obtain the web-service token, login to Silk Central and open the **User Settings**. Here you can also regenerate and delete the token.

   Note: A web-service token is similar to a password and should be protected as such. The token-based authentication offers increased security and convenience, because your Silk Central credentials will not be distributed outside Silk Central and the authentication will not break due to password update policies.

   Note: To use token-based authentication, both Silk Performer 19.5 (or newer) and Silk Central 19.5 (or newer) are required.
6. Click **Test Connection** to verify the URL and the credentials or the token.
7. Click **OK**.

   These are now the default settings for all tasks that require a connection to Silk Central, which includes opening projects from Silk Central, importing projects from Silk Central, and uploading projects to Silk Central.

Opening Silk Performer Projects from Silk Central

Note: If Silk Central is using StarTeam for source-control integration, you need to install StarTeam Microsoft SCC Integration on your computer.

1. Click **File** > **Open Project from Silk Central** . The dialog that displays shows the default settings for connections to Silk Central.These default connection settings can be specified in the **System Settings** on the **Silk Central** page.
2. If you are connecting for the first time, or if you do not want to use the default settings, specify the settings as follows.
3. In the **URL** field, specify the URL of the host on which Silk Central's front-end server is running. Make sure to specify the full URL including the port and the Silk Central instance.
4. Click **Use credentials** and type valid credentials for Silk Central in the fields **Username** and **Password**.
5. Alternatively, click **Use token** and type or paste the web-service token into the field. To obtain the web-service token, login to Silk Central and open the **User Settings**. Here you can also regenerate and delete the token.

   Note: A web-service token is similar to a password and should be protected as such. The token-based authentication offers increased security and convenience, because your Silk Central credentials will not be distributed outside Silk Central and the authentication will not break due to password update policies.

Note: To use token-based authentication, both Silk Performer 19.5 (or newer) and Silk Central 19.5 (or newer) are required.

6. Click **Test Connection** to verify the URL and the credentials or the token.
7. *Optional:* Click **Set as Default** to define the specified connection settings as the default settings for future connections.
8. *Optional:* Click **Internet Options** to define Internet connection properties, such as proxy settings, security settings, and browser connection settings.

   If you perform this step, click **OK** on the **Internet Properties** dialog box to save the settings.
9. On the **Open Project from Silk Central** dialog box, click **Next** to view a list of the Silk Central projects to which the active user has read access.

   This list is the same list that is available in Silk Central's **Projects**'s unit.
10. Select the Silk Central project to open and click **Next**.

    Alternatively, you can double-click the project to open.
11. In the **Project** menu tree, open test containers and test folders as necessary and select the Silk Performer test definition to open in Silk Performer.

    If the Silk Performer test definition is located on a UNC path, click **Finish**. Otherwise, click **Next** to open the **Target Directory Selection** dialog box and specify the location to which the project is copied.

    If you are using source-control integration like Visual SourceSafe for Silk Central, you are presented with a logon screen for your source-control client. Type valid user connection settings and click **OK** to continue.

Silk Performer projects can also be opened directly from Silk Central by clicking **Open Project** on the Silk Central **Properties** page in the **Test Plan** unit. For more information, refer to Silk Central Help.

For information regarding the configuration of source-control integration with Silk Central, refer to Silk Central Help.

Checking Out and Editing Downloaded Projects

After you download a Silk Central project from Silk Central, the project appears in Silk Performer, where you can edit it before uploading it back into Silk Central source control.

If you have configured source-control integration for your Silk Central installation, Silk Performer automatically checks the Silk Performer project file out of the source-control database.

Note: In the Silk Performer menu tree, red check marks identify Silk Performer projects that have been checked out by way of Silk Central source-control integration.

This integration is facilitated by Silk Performer by using Web Services to connect to Silk Central and request source-control link information, such as the connection string, user name, and LTP project file name.

Note: If Silk Central is using StarTeam for source-control integration, you need to install StarTeam Microsoft SCC Integration on your computer.

1. After you complete and save the required changes to the Silk Performer project file, choose **File** > **Close Project** .

   Alternatively, you can right-click the **Project** menu tree and choose **Check In (and Close Project)**.

   A **Message** dialog box opens.
2. Click **Yes** to confirm that you want to check the edited files back into the source-control database. All subsequent executions of this project through Silk Central use the updated project file.

Importing Projects from Silk Central

Importing a project involves downloading a copy of a project from Silk Central and working with it independently of Silk Central. Changes that you make to an imported project do not effect Silk Central.

Only a standard Web connection is necessary to import a Silk Central project. No source-control tool connection or credentials are required.

Importing a file into Silk Performer enables you to perform *attended test runs* independently of Silk Central. Attended test runs differ from tests that are run automatically based on predefined schedules in Silk Central because they can be executed manually in Silk Performer. Test results can subsequently be uploaded back to Silk Central and associated with a test definition.

Even if you have configured source code control integration with Silk Central, imported project files are not checked out of your source control tool because they are saved locally to your hard disk as compressed ZIP archives. Attended tests can also be initiated from the Silk Central side.

1. Choose **File** > **Import Project from Silk Central** . You are presented with the connection settings that are specified on the **System Settings - Workbench - Silk Central** page.
2. *Optional:* If you are connecting for the first time, or if you do not want to use the default connection settings, specify the following information:
    - In the **URL** field, specify the URL of the host on which Silk Central's front-end server is running. Make sure to specify the full URL including the port and the Silk Central instance.

      Note: The port is usually port `80` if you are using an ISAPI Web server or port `19120` if you are using Silk Central's standalone Web server.
    - Type valid user credentials for your Silk Performer installation in the **Username** and **Password** fields.
3. *Optional:* Click **Set as Default** to define the specified connection settings as the default settings for future connections.
4. *Optional:* Click **Internet Options** to define Internet connection properties, such as proxy settings, security settings, and browser connection settings.

   If you perform this step, click **OK** on the **Internet Properties** dialog box to save the settings.
5. On the **Import Project from Silk Central** dialog box, click **Next** to view a list of the Silk Central projects to which the active user has read access.

   This list is the same list that is available in Silk Central's **Projects**'s unit.
6. Select the Silk Central project to import and click **Next**.

   Alternatively, you can double-click the project to import.
7. In the **Project** menu tree, open test containers and test folders as necessary and select the Silk Performer test definition to import to Silk Performer.
8. If you do not want to use the default target directory, type a different directory in the **Target Directory** text box.
9. Click **Finish**.

Uploading Projects to Silk Central

Before uploading a Silk Performer project to Silk Central, ensure that you have properly configured the project's workload through Silk Performer. Workload settings are subsequently specified along with uploaded projects in the Silk Performer **Test Properties** portion of the **Properties** page in Silk Central’s **Tests** unit.

1. In Silk Performer, open the project that you want to upload and choose **File** > **Upload Project to Silk Central** .

   If the project has previously been uploaded to Silk Central, the association with a Silk Central test is already known and preselected. Click **Next** and go to step 3.
2. From the **Projects** list, select the Silk Central project to which you want to upload the Silk Performer project and click **Next**.

   You can select an existing test to replace or you can create a new test by right-clicking the appropriate folder or container choosing **New Child Test**.

You can also create a new folder within an existing test container by right-clicking a container and choosing **New Child Test Folder**.

Note: Newly created tests and folders are displayed with bold text to indicate that they have not yet been written to the Silk Central database. If you click **Cancel**, the items you have created are not saved.

3. Check the **Enable Results Upload** check box to select specific test run results to upload with the project and then click **Next**.

   If you do not want to upload test run results at this time, do not check the **Enable Results Upload** check box. Instead, click **Finish**.
4. On the **Select results** page, check the appropriate check boxes in the **Results** list to select the results that you want to upload along with the project and then click **Next**.
5. Specify version and build numbers for the assigned product to which the uploaded results belong.
6. Click **Finish** to upload the project to Silk Central. The newly uploaded items appear in Silk Central's test and **Project** menu tree.

Uploading Test Results to Silk Central

1. Choose **Results** > **Upload Results to Silk Central** .

   Alternatively, you can right-click a results node and choose **Upload Results to Silk Central** .

   If the project has previously been uploaded to Silk Central, the association with a Silk Central test is already known and preselected. Click **Next** and go to step 4.
2. From the **Projects** list, select the Silk Central project to which you want to upload the Silk Performer test results and click **Next**.
3. From the menu tree, select the test to which you want to upload the results and click **Next**.

   Alternatively you can right-click in the tree and choose various menu items to create a new test, child test, test folder, or child test folder to which the results are saved.
4. On the **Select results** page, check the appropriate check boxes in the **Results** list to select the results to upload along with the project.
5. Click **Next**.
6. On the **Specify product information and result status** page, specify the version and build numbers for the assigned product to which the uploaded results belong.
7. Check the checkbox **Enable automatic test result upload** to let Silk Performer upload test results automatically after each test run.

   Note: You can click the **Upload path** link to start Silk Central and go directly to the selected test. Check the **Open Upload path on finish** check box to automatically perform this action after clicking **Finish**.
8. Click **Finish** to upload the results. A confirmation dialog box asks if you want to delete the local copies of the uploaded results.
9. Click **Yes**.

Downloading Test Results from Silk Central

1. In Silk Performer, click the **Results** tab.
2. Click the **Click here to add Silk Central results** node in the **Results** menu tree.

   To download results from another test definition, project, or Silk Central server, click **Back** to navigate back through Silk Central's menu tree.

   The **Open results from Silk Central** wizard opens. The corresponding Silk Central's test definition is preselected. You can select multiple executions and download the results of multiple test executions.

3. In the **Test executions from the last <> day(s)** text box, specify how many past days' worth of results to retrieve.
4. Click **Finish** to download the results.

Silk Performance Manager Integration

This section describes how to integrate Silk Performer with Silk Performance Manager by creating client monitors and infrastructure monitors and how to upload them to Silk Performance Manager.

Creating a Silk Performance Manager Client Monitor

1. Start the Silk Performer Monitor Workbench or switch to the Monitoring workflow bar by right-clicking the workflow bar and clicking **Show Monitoring Workflow Bar**.
2. By clicking the respective buttons on the workflow bar, create a project, record a transaction, and customize your script as usual.
3. Perform a Try Script run.
4. Click **Upload Project** on the workflow bar.
5. Click **Upload project to Performance Manager**.
6. On the subsequent dialog, specify all necessary settings and click **OK**.

Before a project is uploaded to Silk Performance Manager, Silk Performer automatically sets default threshold values for all measures. The thresholds are calculated either from an existing baseline (if such exists) or from the latest Try Script run.

Creating a Silk Performance Manager Infrastructure Monitor

1. Click **File** > **New Project** or click **Start here** on the workflow bar if no project is open. The **Workflow - Outline Project** dialog opens.
2. Enter a **Name** and optionally a **Description**.
3. From the list of application types, select **Monitoring** > **Performance Manager - Infrastructure Monitor** and click **Next**. Performance Explorer opens and the **Data Source Wizard** displays.
4. Follow the wizard to select a data source. A monitor chart displays and the monitoring automatically starts.
5. On the **Real-Time Monitoring** tab, in the **Export** group, click **As Project**.
6. Follow the **Reuse Monitor Wizard** to export the monitor chart.
7. Specify the settings on the **Upload Project** dialog and click **OK**.

Source Control Integration

Silk Performer's source control integration (SCC integration, also referred to as source code control integration) enables you and other members of your organization to manage and share your Silk Performer project files, test scripts, custom include files, and data files in a common repository (source control system)—enabling file check-in, check-out, get-latest-version, and other functions typical of source control systems.

Silk Performer’s source control integration is facilitated by commands on the Silk Performer **File** menu (**File** > **Source Control**) and context menus in Silk Performer’s **Project** menu tree. All project nodes, script nodes, custom include file nodes, and data file nodes in the **Project** menu tree offer SCC commands.

In addition to offering commands that select all the files of complete projects (project file, custom include files, data files, and scripts), a sub-set of SCC commands that select only the project file are also available (you can easily check-in/check-out project files without overwriting associated script and data files).

Status icons in Silk Performer's **Project** menu tree indicate the status of files under source control:

- Files that have been placed under source control and are currently checked out (the source control version of the file is writable) are indicated with red checkmark icons.
- Files that have been placed under source control, but are currently NOT checked out (the source control version of the file is read-only) are indicated with blue padlock icons.

Silk Performer supports StarTeam, Microsoft Visual SourceSafe, and PVCS. Source control integration must be enabled in Silk Performer system settings (**Settings** > **System** > **Source Control**). You must have StarTeam, PVCS, or Visual SourceSafe installed on your system before source control integration can be enabled.

Integrating StarTeam SCC with Silk Performer

The integration consists of downloading the integration module and running the installation wizard.

Downloading the StarTeam SCC Integration Module

1. Navigate to the StarTeam product downloads area for the latest StarTeam Integration for SCC release at *http://supportline.microfocus.com/websync/StarTeam2009EA.aspx*.

   You may need to login or verify your email address.
2. Scroll down to the **StarTeam 2008 R2 Integration for Microsoft SCC** section near the bottom of the page.
3. Locate and double-click the `ST-SCC-Int-10.4.7-Win-EN.exe` file.
4. Click **Run** or **Save**, depending on which browser you are using.
5. Specify the location where you want to save the SCC Integration files or executable file and then click **Save**.

Follow the steps in the StarTeam SCC Integration Module wizard to install the SCC Integration module.

Running the StarTeam Microsoft SCC Integration Setup Wizard

Download the StarTeam SCC Integration Module before beginning this task.

1. Choose one of the following:

   - Immediately after the download completes, click **Launch** on the **Download Manager** to launch the installation wizard.
   - At your convenience, navigate to the location where you saved the SCC Integration executable file, `scc_integration.exe`, and double-click the file.

   The **SCC Integration InstallShield wizard** opens.
2. Click **Next**. The **Welcome** page opens.
3. Click **Next**. The **Software License Agreement** page opens.
4. Click **Yes** to install the SCC Integration module. The **User Information** page opens.
5. Specify your **Name** and **Company** and then click **Next**.

   The StarTeam SDK runtime is installed if needed.

   The **Choose Destination Location** page opens.
6. Accept the default destination location or click **Browse** to specify another location and then click **Next**. The **Select Components** page opens.
7. Select the **SCC API Integration** check box and then click **Next**. The **Folder Selection** page opens.

8. Accept the default program folder location or specify another folder and then click **Next**. Setup adds program icons to the folder that you specify.
9. Select the language for your installation from the list box and then click **OK**. The **File Compare/Merge** page opens.
10. Click **Next**. The **License Agreement** page opens.
11. Specify whether you agree to accept the license terms and then click **Next**. The **Choose Install Folder** page opens.
12. Accept the default location or click **Choose** to select another location and then click **Next**. The **Choose Shortcut Folder** page opens.
13. Specify where you want to install the shortcut and then click **Next**. The **Pre-Installation Summary** page opens.
14. Click **Install**.
15. Click **Done**. The **Setup Complete** page opens.
16. Specify whether you want to restart your computer now or later and then click **Finish**.

Important: It is critical that you set up and manage your projects using the StarTeam client. This includes creating projects, adding files to a project, and removing files from a project.

Configuring Source Code Control Integration in Silk Performer

Ensure that StarTeam, MS VSS, or PVCS is installed on your local system before beginning this task.

1. Select **Settings** > **System** . The **System Settings** dialog opens.
2. Select the **Workbench** group icon.
3. Click the **Source Control** tab (this tab is farthest to the right of the tab list; use the small right-pointing arrow to access this tab).
4. Check the **Enable Source Control integration** check box. All SCC providers that are installed on your local machine and supported by Silk Performer (i.e., StarTeam, VSS, or PVCS) are listed in the **SCC Provider** list box.
5. Enter a username in the **Username** field.

   The username you enter here will be suggested automatically as the default username value in future operations, such as login dialogs.

   Note: Custom include and user data files are configured independently from project and system files for the SCC because Silk Performer may require that these files be stored separately in the SCC (e.g., an organization might have a common repository for include files that are shared amongst a group of developers. These developers might also be required to check in files from their individual projects).
6. In the **Custom user data files SCC path** field, click **[...]** and select the location in the SCC database where the custom user data files are to be saved.
7. At this point you will be required to log into your SCC system.

   The username that you entered above will be entered into the login dialog automatically.

   See StarTeam, MS VSS, and PVCS documentation for full details regarding the use of your source control system.

   a) Your SCC system will also likely require that you enter the password that corresponds to the submitted username.
   b) If not already selected, you may need to select the target SCC database.
   c) Following that, browse the file structure of your SCC system and select the directory to which your custom user data files are to be saved (you may also create a new directory for this purpose).

   The **SCC database path** field is automatically populated with the database-path root of the source control system you specified.

8. In the **Custom include files SCC path** field, click **[...]** and select the location in the SCC database where the custom include files are to be saved.
9. Repeat step 7.

   The **SCC database path** field is automatically populated with the database-path root of the source control system you previously specified.
10. Click **OK** to exit the **System Settings** dialog. Source control integration is now complete. Source control functions are now available from the File menu (**File** > **Source Control**).

Placing Projects and Files Under Source Control

Place a project and all its associated files under source control or place an individual file under source control.

Placing a Complete Project Under Source Control

Note: System include files and data files cannot be placed under source control. Only custom include files and data files can be placed under source control.

1. With a project loaded into Silk Performer, select the project node in the **Project** menu tree.
2. Choose one of the following:

   - Select **File** > **Source Control** > **Add to Source Control** .
   - Right-click the project node and select **Add to Source Control** from the context menu.

   The **Add To Source Control** dialog opens. All scripts, include files, and data files associated with the project are automatically selected for placement under source code control.
3. Uncheck any file type selections for file types that should not be placed under source control.
4. Ensure that source control paths have been configured for each project file type. As you have already configured source control paths for custom include files and data files in Silk Performer's system settings, you should now only have to specify source control paths for project files and script files. If required, log into your SCC and specify directories for your project and script files.
   For example, with VSS, a **Click here to specify** link appears in the **Source Control Project** column of the **Add to Source Control** dialog for any project file type for which a source control path has not been specified.
5. Click **Add** to place the project's files under source control.

Placing Individual Files Under Source Control

Note: System include files and data files cannot be placed under source control. Only custom include files and data files can be placed under source control.

Note: Project files (`.ltp`), script files (`.bdh`), custom include files, and custom data files can be independently placed under source code control.

1. In the **Project** menu tree, choose one of the following:

   - Right-click a file and select **Add to Source Control** from the context menu.
   - Right-click the **Scripts**, **Data Files**, or **Include Files** nodes and select **Add to Source Control** from the context menu to select all files of those types for placement under source control.

   The **Add to Source Control** dialog opens, showing the files upon which the operation is to be performed.

   Note: The remainder of this process is handled via dialogs provided by your SCC provider (StarTeam, PVCS, or VSS). See StarTeam, MS VSS, and PVCS documentation for full details regarding the use of your source control system.
2. Select the check boxes for the files that you want to place into source control.

Note: If not already defined, a source control project path must be defined for each file that is to be placed under source control. If a file marked for placement under source control does not yet have a source control project path configured for it, you will be prompted to configure a path (the source control project of the Silk Performer project file (`*.ltp`) is suggested by default).

Note: Files added to source control are written to the corresponding `*.ltp` file. Therefore, to add files to source control, the corresponding `*.ltp` file must be checked out.

3. Click **Add** to place the file under source control. A blue padlock appears next to files in the **Project** menu tree that are under source control and are currently checked in (read-only). This is the default status of files that are newly placed under source control.

Checking In Files

1. In the **Project** menu tree, perform one of the following steps:
   - To check in an individual file, right-click the file that is currently checked out, as indicated by a red check mark, and choose **Check In**.
   - To check in all the files of a specific type, right-click the **Scripts**, **Data Files**, or **Include Files** node and choose **Check In**.
   - To check in an entire project, right-click the project node and choose **Check In**.

   The **Check In** dialog box opens, showing the local path where the file or project is currently saved.
2. *Optional:* Deselect any files that you do not want to check in.
3. Click **Check In** to check in the file or project. The status icon of the file appears as a blue padlock, indicating that the source control version of the file or project is now read-only.

Checking Out Files

1. In the **Project** menu tree, choose one of the following:
   - To check out an individual file, right-click the file that is currently checked in (indicated by a blue padlock icon) that you want to check out.
   - Right-click the **Scripts**, **Data Files**, or **Include Files** nodes to select all files of those types for check out.
   - To check out an entire project, right-click the project node.
2. Select **Check Out** from the context menu. The **Check Out** dialog opens, showing the local path to where the file or project will be copied.

   Note: If a single checked-in file is selected for check out, then only that file will be listed on the **Check Out** dialog. If no file or a checked-out file is selected for check out, then all currently checked-in files of the project will be listed on the **Check Out** dialog.
3. If required, deselect individual files that you do not want to check out.
4. Click the **Check Out** button to check out the files or project. The status icon of the files now appear as a red checkmark, indicating that the source control version of the files or project are available and writable.

Undoing File Check Out

1. In the **Project** menu tree, choose one of the following:
   - To undo checkout of an individual file, right-click a file that is checked out (indicated by a red checkmark icon).
   - Right-click the **Scripts**, **Data Files**, or **Include Files** nodes to select all files of those types at once.
   - To undo checkout of an entire project, right-click the project node.
2. Select **Undo Checkout** from the context menu. The **Undo Checkout** dialog opens, showing the local path to where the file or project is located.

3. If required, deselect individual files that you do not want to check out.
4. Click the **Undo** button to undo the check out. The status icon of the file or project now appears as a blue padlock, indicating that the source control version of the file is read-only.

Getting the Latest Version of a File from the Source Control System

These instructions apply to the **Get Latest Version** command for individual scripts, include files, data files, and entire projects.

Note: See StarTeam, MS VSS, or PVCS documentation for full details regarding use of your SCC's **Get Latest Version** command.

1. In the **Project** menu tree, choose one of the following:
   - To get an individual file, right-click the file in the **Project** menu tree.
   - Right-click the **Scripts**, **Data Files**, or **Include Files** nodes to select all files of those types.
   - To get the latest version of an entire project, right-click the project node.
2. Select **Get Latest Version** from the context menu. The **Get Latest Version** dialog opens, showing the local path where the file will be copied.
3. If required, deselect individual files that you do not want to get.
4. Click the **Get** button.

   Note: Get Latest Version commands do not change the checked-in/checked-out status of files. For example, if the **Get Latest Version** operation is performed on a file that is currently checked out, the file remains checked out (if the file is open, you will be prompted as to whether or not you want to have the file reloaded with the new version).

Project-File Only Source Control Commands

In addition to offering source control commands that operate on all the files of a complete project (project file, custom include files, data files, and scripts), a sub-set of SCC commands that select only the project file itself are available from the File menu (**File** > **Source Control** > **Project File Only**) and the project node of the **Project** menu tree. These commands operate the same as their counterparts for associated files do; the only difference is that they do not affect the associated files (scripts, custom include files, and data files). Such functionality is useful if, for example, you want to check in a project file without overwriting the associated script, custom include, and data files that are currently stored in the SCC.

The project-file only commands that are available from the File menu (**File** > **Source Control** > **Project File Only**) and the project node of the **Project** menu tree include:

- **Get Latest Version**
- **Check Out**
- **Check In**
- **Undo Checkout**
- **Add to Source Control**

Removing Source Control from a Project

The **Remove Source Control Linkage** command is the reverse of the **Add to Source Control** command. This command is useful when working with files (possibly received from a customer or support staff) for which the associated source control system is not available.

Note: Source control linkage can be removed for complete projects only, not individual files.

1. Choose one of the following:
   - In the **Project** menu tree, right-click the project node of a project that is under source control and select **Remove Source Control Linkage**.
   - Select **File** > **Source Control** > **Remove Source Control Linkage** .
2. Click **OK** to confirm that you understand that all source control connection data for the project will be lost.

Synchronizing the Source Control Status of a Project

Under certain conditions, Silk Performer may fall out of sync with source control systems due to the corruption of linkage information. For example, if one were to close Silk Performer while Silk Performer is loaded with a checked out file and then check in that same file via the SCC, Silk Performer would not be aware that the file had been checked in.

Note: This function can be applied to complete projects only, not individual files.

1. Choose one of the following:
   - In the **Project** menu tree, right-click the project node of a project that is under source control and select **Synchronize**.
   - Select **File** > **Source Control** > **Synchronize** .
2. Click **OK** to confirm that you understand that all source control connection data for the project will be lost.

Source Control Known Issues and Workarounds

The following issues may occur when using source control.

SCC Client's Reconnect Functionality Is Not Supported

Issue	Workaround
When Silk Performer projects and/or files are linked to a defined SCC directory, and the directory is changed via the SCC client, Silk Performer does not reconnect to the new directory.	Remove the existing source control linkage then add the affected projects/files to the new directory.

Note: It is generally not recommended to use SCC client functionality for maintaining linked Silk Performer projects.

MS VSS 8.0 Requests Username/Password for Each File Check-in/ Check-out Operation

Issue	Workaround
MS VSS 8.0 introduced a security setting that, when activated, requests the user to enter their username and password each time they check in or check out a file from VSS. This request also appears in Silk Performer when performing file operations with VSS.	Disable the MS VSS manual login.

Disabling the MS VSS Manual Login

Note: This procedure only works if the username in VSS is identical to the network username.

1. Open **SourceSafe Administrator**.

2. Open the database where your Silk Performer projects are saved.
3. Select **Tools** > **Options** .
4. Click the **General** tab.
5. Check the **Use network name for automatic user login** check box.

Importing, Uploading, and Emailing Projects

Silk Performer enables you to export, import, email, and upload projects.

Silk Performer offers the option of exporting entire test projects to a separate directory or an archive file. Such exported project files can subsequently be imported into Silk Performer.

Exporting Projects

1. From the Silk Performer menu bar, select **File** > **Export Project** . The **Export Project** dialog appears.
2. In the **Export location** area, specify the location to which you want to export the current project.

   If you want to export all the project files to a directory, specify the directory name. If you want to export all the files to a ZIP archive, specify the archive name. Click **[...]** to the right of the field to locate the directory or file where the project is to be exported.
3. Select the **ZIP to single archive file** option to export all files that belong to your current project to a ZIP archive.

   This is useful when you want to backup your project or when you want to send a copy of your project to someone.
4. Select the **Open in Windows Explorer after export** check box to have the folder in the export location open automatically in Windows Explorer after an export.

   This provides you with quick access to the project file after the export, without having to navigate to find the directory where the file is stored.
5. In the **Options** area, select the **Include results files of all tests** option to export the results of all tests that belong to this project (the results of all TryScript tests, baseline tests, and actual tests).

   If you disable this option, only the results of the most recent TryScript test will be exported.
6. Select the **Protect ZIP file with password** option to protect the export file (LTZ) with a password.

   This is useful when your project contains confidential data (for example, user names and passwords for the application you are testing). If you select this option, enter a password and confirm it.
7. Select the **Include a system diagnostic report** check box to include a diagnostic report with the export.

   This report lists the processors in the computer, the name and version of the operating system, and information regarding the drives, memory, services, and drivers. Relevant information from the Windows registry is also included.
8. Click **OK**. Silk Performer then exports the project and displays the progress of packing all relevant files into a ZIP archive in a status bar.

Importing Projects

1. From the Silk Performer menu bar, select **File** > **Import Project** .

   Note: If you have a project open, you will be prompted to close your project. Click **Yes**.

   An **Open** dialog appears. Browse to and select the archive file that you want to import.
2. After selecting the archive file, click **OK**.
3. The **Import Project** dialog appears.

4. In the **Import target directory** area, specify the location where you want to extract the project files.
   a) If you have configured password protection, you will be prompted to enter your password. Enter your password and confirm it.
5. Click **OK**. Silk Performer then extracts the archive file to the specified target directory and opens the project.

Emailing Projects

1. From the Silk Performer menu bar, choose **File** > **Email Project** . The **Email Project** dialog box opens.
2. In the **Email address** field, specify the email address where your project is to be sent.
3. In the **Options** area, check the **Include results files of all tests** check box to attach the results of all TryScript tests, baseline tests, and actual tests that belong to this project.

   If you uncheck this check box, only the results of the most recent TryScript test are sent.
4. Check the **Protect archive file with password** check box to protect the contents of the email with a password.

   This feature is useful when your project contains confidential data, such as user names and passwords for an application under test. If you check this check box, type a password and confirm it.

   Note: Inform the recipient of the password in a phone call or separate email.
5. Check the **Include a system diagnostic report** check box to attach a system diagnostic report.

   This report includes the following information:

   - Computer processors
   - OS name and version
   - Information regarding the drives, memory, services, and drivers
   - Relevant information from the Windows registry
6. Click **OK**. Silk Performer emails the project, using a status bar to display the progress of packing all relevant files into a ZIP archive.

Troubleshooting

All customers whose products qualify for maintenance and all prospective customers who are evaluating products are eligible for Technical Support.

Results Recovery Workflow

In environments with unreliable network connections such as the Internet, unforeseen exceptions can happen. If the controller loses connection to an agent during a load test or while collecting results, Silk Performer will try to re-establish the connection. However, this might not always be possible and would lead to an incomplete results set.

For such a case, Silk Performer provides a recovery workflow. It allows you to recover results and build a new set of valid data. Recovering results is not an ideal solution, but it prevents you from entirely losing the already gathered data.

The recovery workflow varies, depending on the type of load test you have executed.

Recovering CloudBurst load tests

Assumed, you are executing a CloudBurst load test. For an unknown reason, the Workbench becomes unresponsive and finally crashes. When you restart the Workbench, the recovery workflow dialog automatically displays. Follow the recovery workflow and review the following fields, which are automatically prefilled:

- **Missing results**: This folder contains the results that could not be completely transferred from the agents to the controller. The transfer of these results was interrupted by the crash and therefore the data was truncated.
- **Transferred results**: This folder contains the results that have been completely transferred from the agents to the controller. The transfer of these results was successful and the data is complete and valid.
- **Recovered results**: This folder will contain the recovered results, once the recovery is completed. Silk Performer will merge the missing results and the transferred results and generate the recovered results out of it.

For CloudBurst load tests, these fields are automatically prefilled. Review the directories and adjust them if necessary

Recovering on-premise load tests

For recovering an on-premise load test, we assume you are executing a load test with agents in your server lab. For some reason the connection to several agents gets lost and cannot be re-established. Thus, the load test ends with an incomplete results set. In this case, click **Tools** > **Recover Load Test Results**.

Follow the recovery workflow and specify all required directories. The directories are the same as described above. However, the fields are not automatically prefilled. Before you specify a directory for the missing results, make sure to create this directory on the controller and to create a subfolder for every lost agent. Then, copy the results from the agents into the subfolders. Specify all three directories and start the recovery workflow.

Recovering results

Silk Performer will execute several steps to recover the results. If an issue occurs during recovery, the process stops and you can view a log file for more information. Depending on the issue, you can ignore it and let the recovery process continue, or you can restart the process.

Once the recovery is completed, the new set of results displays in the **Results** tree.

Technical Support

All customers who are under a maintenance and support contract, as well as prospective customers who are evaluating products are eligible for technical support. Our highly trained customer care staff will respond to your requests as quickly and professionally as possible.

If you have a question on recording Web traffic, load testing scenarios or any other product or implementation issue, our highly trained technical support staff will respond to your requests as quickly and professionally as possible.

To reach Micro Focus SupportLine, visit *https://supportline.microfocus.com*. First time users may be required to register to the site.

Please have the following information ready when you call:

- Product name and version number
- Operating system name and version
- Exact wording of any error messages involved
- Customer ID found on your shipping receipt for Silk Performer or on an index card found in your software box

Controller Agent Communication Troubleshooting

The following issues may occur when a controller computer attempts to connect to agent computers.

Load Test Controller - 3211: Could not connect agent control service

Possible Cause	Resolution
The agent control service is not running.	Start the agent control service.
A port mismatch occurs (the port that the agent control service is listening to and the port at which the controller wants to connect the agent control service are different).	Adjust the agent connection settings.
The agent is connected through a proxy and the proxy is down.	Check the agent connection settings and restart the proxy.

Load Test Controller - 3212: Could not authenticate the Agent Control Service

Possible Cause	Resolution
The agent requires a password and encryption is enabled. The agent's server certificate could not be found or a required dynamic link library (`.dll`) is missing (for example "`ssleay32.dll`"). However, note that the agent control service does not report an error when it is started, if this is the problem.	Your best option is to re-install the remote agent, because missing certificates or dynamic link libraries are indications that the agent was not installed correctly.

Load Test Controller - 3213: The agent control service could not create the agent

Possible Cause	Resolution
This is the same type of error as 3212, except that the agent does not require a password.	See "Load Test Controller - 3212: Could not authenticate the Agent Control" Service"

Load Test Controller - 3223: Agent connection lost

Possible Cause	Resolution
This occurs when the connection is broken between the controller and agent or when the underlying communication units of the load test controller report connection problems. Furthermore, connecting the agent through a proxy might be the cause of the error if the proxy itself is buggy.	There should be log files named `orbClient_<agentName>.log`, `orbServer_<agentName>.log` in the controller's or agent's home directory, respectively, where logging information related to the error can be found.

Load Test Controller - 3233: The agent's version does not match the version of its controller

Possible Cause	Resolution
The agent's version does not match the version of its controller.	Install the correct remote agent version.

Load Test Controller - 3303: The local results directory could not be created

Possible Cause	Resolution
This is an access rights problem, that is, the user account of the agent process does not have the necessary access rights to create a directory; the parent directory of the desired directory is probably a read-only version for all accounts.	Grant the necessary access rights.

Load Test Controller - 3304: The local results directory could not be cleared

Possible Cause	Resolution
A file within the local results directory is still open and being used by another process (most likely the `perfLtcAgent.exe` and `perfRun.exe`). This is usually a hint that either the agent or the runtime processes did not shutdown at the end of a previous load test, which should not happen.	Check for pending `perfLtcAgent.exe` or `perfRun.exe` files and terminate them manually. If this is not the case, check that other programs are not using an open file from the local results directory. If none of the above options solve the problem, reboot the remote agent's machine.

Load Test Controller - 3305: The local data directory could not be created

Possible Cause	Resolution
See "Load Test Controller - 3303: The local results directory could not be created"	See "Load Test Controller - 3303: The local results directory could not be created"

Load Test Controller - 3306: The local data directory could not be cleared

Possible Cause	Resolution
See "Load Test Controller - 3304: The local results directory could not be cleared".	See "Load Test Controller - 3304: The local results directory could not be cleared".

Load Test Controller - 3422: Command (connection) timed out

Possible Cause	Resolution
Usually the controller has 30 seconds to: • Connect to the remote agent control service • Pass authentication information to the remote agent control service (if required) • Ask the agent control service to create the agent • Connect to the agent • Exchange version information with the remote agent	Specify a higher timeout in the system settings.

Possible Cause	Resolution
These tasks may take more time than the default timeout of 30 seconds, especially if the agent is connected through a proxy or if the connection is slow.	

Silk Performer - 32: Access denied, authentication required

Possible Cause	Resolution
The agent expects a password, but the controller did not send a password, meaning the controller did not authenticate itself.	Adjust the agent connection settings.

Silk Performer - 33: Authentication failed (wrong password)

Possible Cause	Resolution
The agent expects a password but the password sent by the controller is incorrect.	Adjust the agent connection settings.

Silk Performer - 25: Limit of concurrent instances exceeded

Possible Cause	Resolution
Two agents cannot run simultaneously on the same computer. Most likely, the agent is involved in a load test started by another controller.	Wait for the other controller to finish its load test and then try again. If this is not the problem, the agent did not terminate correctly at the end of a previous load test and you must terminate the agent process manually.

System - 2: The system cannot find the file specified

Possible Cause	Resolution
The agent executable (`perfLtcAgent.exe`) could not be found or there is a different entry for the agent executable in the `performerLocal.ini` file.	Check the `performerLocal.ini` file. If it is not there or if the `perfLtcAgent.exe` is not present, then re-install the agent.

Error Message Overview

Silk Performer system-related errors occur either during script compilation or during load test execution and can cause load tests to fail. As they occur, Silk Performer reports system-level errors on the screen. Severe system-level errors are reported to the Windows NT Event Log.

Silk Performer distinguishes among the following types of system-level errors:

- Compiler errors
    - Lexical errors
    - Semantic errors
    - Syntax errors
- Internal errors
- System limitation errors
- Runtime errors

Each error message is identified by a sequential error number (unique within an error category) plus a three-character code that indicates the category of error, followed by a string providing a detailed description of the error.

Compiler Errors

If an error occurs during compilation, the Silk Performer compiler stops at the position where the error was detected. The error message is displayed, and the line that caused the error is highlighted.

The compiler distinguishes among the following categories of errors:

- Lexical errors (LEX)
- Semantic errors (SEM)
- Syntax errors (SYN)

Lexical Errors

A lexical error (error type LEX) occurs if the compiler detects lexical symbols that are not allowed in Benchmark Description Language.

Example

If a script contains an invalid character such as '{', compilation is aborted and an error message is printed:

```
benchmark WebTest

dcluser
  user    {
```

Output

```
compile error LEX 13: not a lexical symbol of BDL
--- compilation was not successful ---
```

Semantic Errors

A semantic error (error type SEM) occurs if semantic conditions do not comply with the requirements of Benchmark Description Language.

Example

If you use an undeclared variable ("i" in the example below), compilation is aborted and an error message is displayed:

```
dcltrans
  transaction TMain
  var
    artname      : string(40);
    artno, price : number;
    result set c1;
  begin
    writeln; write("transaction TMain:"); writeln;
    c1: SelArticle(out artno, price, artname);
    write("rows found: "); write(rows(c1));
    fetch c1 all;
    for i:
```

Output

```
compile error SEM 12: variable is not declared
--- compilation was not successful ---
```

Syntax Errors

A syntax error (error type SYN) occurs if the syntax of a test script does not agree with the syntax of Benchmark Description Language.

Example

If you do not specify the keyword `then` within an `if` statement, an error message appears:

```
transaction TMain
var
  j : number init 1;
  i : array [500] of number;
  s : array [5] of string(10);
begin
  if j=1
  j:
```

Output

```
compile error SYN 38: 'then' expected
--- compilation was not successful ---
```

Internal Errors

Silk Performer can detect internal script errors (error type COM). Please contact Customer Care if you encounter one.

System Limitation Errors

System limitation errors (error types RES) can occur while either compiling a script or running a load test . If they occur during a test execution, they will cause the test to fail. These errors indicate that a system limitation has been exceeded.

Runtime Errors

Only the Silk Performer simulator will produce runtime errors (error type RUN). Runtime errors indicate programming errors in test scripts that cannot be recognized during compilation (for example, using array indexes outside the scope of an array). If a runtime error occurs, an error message is printed on screen and to a `.rpt` file.

Known Issues in Silk Performer

This list contains known issues in Silk Performer and provides work-arounds where available:

General Silk Performer Issues

Multi-byte character set support limitations

Problem:

The protocol / testing support of the Multi-Byte Character Set / UTF-8 / EUC-JP is limited to certain application types.

Resolution:

The protocol / testing support of the Multi-Byte Character Set / UTF-8 / EUC-JP is limited to the following application types:

- Web application testing, both protocol-level and browser-driven level
- SAPGUI testing
- Oracle Forms/Oracle Applications testing
- Citrix testing

.

How can I get Silk Performer "What's This" contextual Help to work on Windows Vista or later?

Problem:

How can I get Silk Performer "What's This" contextual Help to work on Windows Vista or later Windows versions?

Resolution:

In Silk Performer it is possible to get information about GUI functionality by right-clicking GUI controls and choosing **What's This?**. This contextual help does not work on Windows Vista or later Windows versions. This is a known issue related to the fact that the Windows Help program `WinHlp32.exe` is no longer included with newer Windows versions. To resolve this, download the respective .exe file from the Microsoft Download Center. For more information, see: *http://support.microsoft.com/kb/917607*.

The Silk Launcher Service is falsely removed when a previous version of Silk Performer is removed

Problem:

If you have multiple versions of Silk Performer installed on your system and you remove a previous version of Silk Performer, the Silk Launcher Service is falsely also removed.

Resolution:

Uninstall all Silk Performer versions from your system and reinstall the needed versions in ascending order.

GUI-level testing does not work if Silk Test is not installed in the Program Files folder

Problem:

GUI-level testing does not work if Silk Test is not installed in the `Program Files` (`Program Files (x86)`) folder.

Resolution:

None.

GUI-level scripts generated from Japanese libraries do not compile

Problem:

When importing Silk Test test cases with names that contain non-ASCII characters, the generated script stubs cannot be compiled.

Resolution:

Either rename the test case in Silk Test so it does not use any non-ASCII characters, or rename the generated transactions in the Silk Performer script accordingly.

The System Configuration Manager entry in the Tools menu is disabled

Problem:

If you have multiple versions of Silk Performer installed on your system and you uninstall a previous version of Silk Performer, the **System Configuration Manager** entry in the **Tools** menu might be disabled.

Resolution:

Re-install the current Silk Performer version.

Reports created from recovered results show incorrect values

Problem:

The test summary, detailed, and region summary report created from recovered results show incorrect values.

Resolution:

Due to limitations in the recovery process, not all values can be recovered completely. The following issues can occur:

- The summary data is missing.
- The number of **Active Users** is incorrect.
- The **Session Time** and the **Session Busy Time** cannot be recovered, they always displayed as `0`.
- The **Average Action Time** cannot be recovered, it always displays as `-`.
- The number of **Errors** is incorrect.

GUI-level testing with keyword-driven tests and Selenium is only supported with Java 8

Problem:

GUI-level testing with keyword-driven tests and Selenium is only supported with Java 8.

Resolution:

This is a Selenium restriction. The only workaround is to use Java 8.

TrueLog Explorer Issues

On-access virus scanner software may cause TrueLog Explorer to crash

Problem:

On-access virus scanner software may cause TrueLog Explorer to crash.

Resolution:

Virus scanner software may cause a variety of problems including TrueLog Explorer crashes or performance degradation. We recommend disabling virus scanner software on the agent computers and on the controller computer for the duration of load tests.

Visual script modifications fail if the related transaction is in an include file

Problem:

Visual script modifications fail if the related transaction is in an include file.

Resolution:

Do not move transactions to BDH files.

Visual user data customization does not work for browser-level scripts

Problem:

Visual user data customization does not work for browser-level scripts

Resolution:

This behavior is by design, as TrueLog Explorer only considers forms submitted by `WebPageSubmit` calls (page-based browser-level API), because the HTML form names are required to match HTML and BDL forms. If you want to customize your script using TrueLog Explorer Visual User Data Customization, you need to record a page-based browser-level API script.

SAPGUI Issues

Control position information cannot be retrieved correctly from SAP

Problem:

In rare instances, control position information can not be retrieved correctly from SAP. Therefore TrueLog Explorer might not be able to display the selected control rectangle at the correct position on the screenshot.

Resolution:

Use the control tree instead of the screenshot.

Citrix Issues

Connecting to existing session during Citrix recording results in unusable script

Problem:

Connecting to existing session during Citrix recording results in unusable script

Resolution:

When connecting to an existing Citrix session, the Citrix client does not recognize any windows and therefore no synchronization functions are scripted. You should always record a new session.

Limitations for load testing with Citrix Receiver 4.4, 4.10 and newer

Problem:

The BDL function `CitrixConnect` does not work for Citrix load testing in Citrix Receiver 4.4, 4.10 and newer. This is due to a defect in these Citrix versions.

Resolution:

Option 1: Use Citrix Receiver (LTSR) 4.9.7 or later.

Option 2: Use the latest Citrix Workspace-App.

Option 3: Use `CitrixConnectIcaData` or `CitrixConnectIcaFile` instead of `CitrixConnect`.

Citrix passthrough authentication is not enabled for custom tools

Problem

The Citrix passthrough authentication mode is not enabled for custom tools. Therefore, Silk Performer cannot record or replay this authentication mode.

Resolution

Recording and replaying Citrix with enabled passthrough authentication is possible and no issues will occur, however you are prompted to enter your credentials anyway. Be aware that though passthrough authentication is enabled, it is in fact not working.

Citrix process is already running under a different user account

Problem

When executing a Citrix load test, the following error message displays: `Citrix process is already running under a different user account`. This happens when the Citrix process `wfcrun32.exe` is already running under a certain user account. For example: A Windows session is already used by the user A. When a load test is started, an agent logs into the same Windows session as user B. User B now attempts to start the process `wfcrun32.exe`, which does not work and causes the error to display in Silk Performer.

Resolution

Close the running Windows session by logging out user A before you start the load test.

Oracle Forms Issues

After hooking into main Oracle Forms applet classes, JInitiator cannot load additional jar files that rely on those classes

Problem

After hooking into main Oracle Forms applet classes, JInitiator cannot load additional jar files that rely on those classes

Resolution

When recording Oracle Forms or Oracle Applications, define the following Java Runtime Options on the **JInitiator Properties** Dialog: `"-noverify -mx128m"` `NoVerify` prevents these loading issues. Additionally, the default **Virtual Memory Size** of JInitiator 1.1.7.x is between 16 and 64 MB. The option `-mx128m` increases the virtual memory size to 128MB. This additional memory is needed as there is some extra memory overhead caused by recording. If you run into an `OutOfMemory` error you can further increase this value.

Browser crashes during recording of Oracle Forms

Problem

Browser crashes during recording of Oracle Forms.

Resolution

When recording Oracle Forms 6i it is recommended that you disable Java JIT (Just-In-Time compiling) in the Java Runtime settings. This is because the JIT library of the Java Virtual Machine (symcjit.dll) may crash your browser.

Java JIT can be disabled using one of the following options:

Option 1: Define `- DJAVA.COMPILER=NONE` in the Java runtime settings of the Oracle **JInitiator Properties** dialog. This must be done using the JInitiator control panel, which is installed with JInitiator (available in the Windows control panel).

1. Launch the JInitiator Control Panel.
2. Add `DJAVA.COMPILER=NONE` to the **Java Run Time Parameters** text field.
3. Click **Apply**.

Option 2: Use the check box on the Oracle JInitiator Properties dialog to disable the Just In Time Compiler (only available since version 1.1.8.x).

1. Launch the JInitiator Control Panel.
2. Select the **Advanced** tab.
3. Uncheck the **Enable Just In Time Compiler** checkbox.
4. Click **Apply**.

Option 3: Define environment variable: `JAVA_COMPILER=NONE`.

1. Right-click **My Computer** and choose **Properties**.
2. Choose **Advanced**.
3. Choose **Environment Variables**.
4. Under **System Variables** click **New**.
5. For **Variable Name** enter `JAVA_COMPILER`.

6. For **Variable Value** enter `NONE`.
7. Click **OK** to exit out of dialogs.

Oracle Forms cannot be recorded

Problem

When attempting to record Oracle Forms, a dialog appears, containing the following question: `Do you want to run this application`? However, the **Run** button on this dialog cannot be clicked and therefore the recording cannot be continued. This happens primarily under Windows 7.

Resolution

Navigate to the `Essentials` folder within your Silk Performer installation directory. Your path should resemble this one: `C:\Program Files (x86)\Silk\Silk Performer <version>\Essentials`. Locate the respective Oracle Forms Essentials file: `OraForms<version>.sep`. The .sep file contains the file `DeploymentRuleSet.jar`. Extract the .sep file and copy `DeploymentRuleSet.jar` to the following directory: `C:\Windows\Sun\Java\Deployment`. Restart Silk Performer. Now, when you start a recording, the dialog no longer appears. `DeploymentRuleSet.jar` is available for Silk Performer 18.5 and newer versions.

Browser-Driven Load Testing Issues

ShowModalDialog and ShowModelessDialog not fully supported

Problem

The non-standard functions `showModalDialog` and `showModelessDialog` are not fully supported. The browser-driven implementation of these functions is not fully compatible with the implementation of Internet Explorer. In particular javascript-heavy pages that are loaded in such a dialog may not function correctly.

Resolution

Try to use a GUI-level testing approach in such situations.

Webbrowser control of primary BrowserWindow is terminated

Problem

Silk Performer's browser-driven approach relies on the webbrowser control of the primary (initial) BrowserWindow through the entire scenario of a test. To implement a particular workflow, some applications open a new child window and close the base window. If the closed window is the primary (initial) BrowserWindow of the test scenario, the internal state of Silk Performer' s browser-driven engine is undefined. This might result in unexpected application behavior during recording as well as during playback.

Resolution

Add an additional BrowserWindow to your test - it will act as the new base window. The primary (initial) BrowserWindow remains unused in your scenario. To add an additional window during recording, click **File** > **New Tab**. Make sure to add the window right at the beginning of your recording. For existing scripts, add a `BrowserCreateWindow` statement right before the first `BrowserGetActiveWindow` statement.

TrueLog Explorer displays the size of uncompressed data in browser-driven tests

When receiving compressed data during a browser-driven test, the results displayed in TrueLog Explorer and in the **Virtual User Report** can be misleading. TrueLog Explorer displays the size of the uncompressed data for the measure **Response data received** in the **Statistics** tab, while the Virtual User Report shows the value `0`. Note that this is only the case for browser-driven tests and when compressed data is transmitted during the test.

Java over HTTP Issues

Previous Java over HTTP scripts might not run successfully

Problem

Java over HTTP scripts recorded with a previous Silk Performer version might no longer run successfully.

Resolution

Due to the upgrade of the third-party component XStream, some Java over HTTP scripts that have been recorded with previous versions of Silk Performer might no longer run successfully. In such a case, re-record your scripts with the current Silk Performer version.

Index