Borland®

Silk TrueLog Explorer 16.0

Help

Borland Software Corporation
700 King Farm Blvd, Suite 400
Rockville, MD 20850

2015-02-11

Contents

TrueLog Explorer 16.0

Welcome to TrueLog Explorer 16.0.

TrueLog Explorer supports Silk Performer testing efforts with a framework from which you can customize test scripts and perform exhaustive analysis of returned test results.

With an array of tools and reports, TrueLog Explorer helps you build automatic content verification into scripts, insert parsing functions for static session data, and gain insight into the root causes of errors via TrueLog On Error functionality.

Getting Started

This section provides overview information regarding TrueLog Explorer, TrueLog files, analyzing tests, and the sample applications.

TrueLog Explorer Overview

TrueLog Explorer supports Silk Performer testing efforts with a framework from which you can customize test scripts and perform exhaustive analysis of returned test results.

With an array of tools and reports, TrueLog Explorer helps you build automatic content verification into scripts, insert parsing functions for static session data, and gain insight into the root causes of errors via TrueLog On Error functionality.

Built-in load testing methodology

TrueLog Explorer has been designed around a best-practice approach to load-test script customization and error analysis that guides you from session-handling customization and content verification to user-data customization and, finally, to error analysis. TrueLog Explorer's workflow bar guides you through this methodology step by step, enabling you to augment tests with session handling, verification checks, and parameterized input data. Ultimately, the workflow bar enables you to run TrueLog On Error analysis.

Note: Most of the examples included in this help system relate specifically to Web applications. However, many of the illustrated concepts can be applied to database, Citrix, Oracle Forms, SAPGUI, AJAX-enhanced Web, terminal emulation-screen applications, and XML-based applications. Many of the concepts can also be applied to the testing of UDP- and TCP/IP-based applications, such as POP3, SMTP, and custom protocols. TrueLog analysis is available for these applications, but script customization is not.

What You Can Do with TrueLog Explorer

TrueLog Explorer enables you to perform the following tasks:

- **Find replay errors quickly and easily** – Using TrueLog compare mode, you can automatically detect differences between record and replay sessions to determine whether your application is functioning as intended. Both record and replay sessions are rendered visually so you can compare differences side by side and view errors as they appear to the user.
- **Customize session handling** – TrueLog Explorer parsing functions allow you to replace static session IDs in scripts with dynamic session IDs, thereby maintaining state information for successful load-test runs. These functions are available for Web, database, and XML applications.

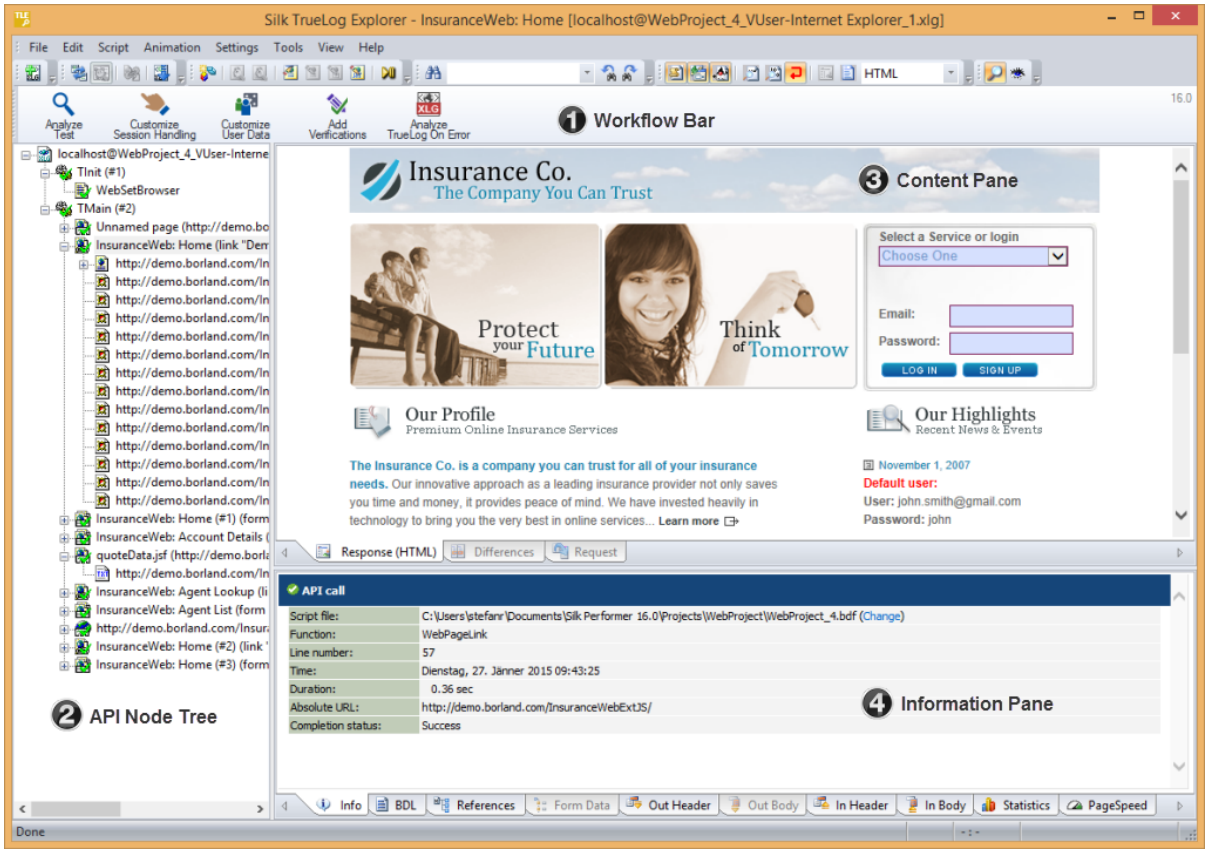
Note: Silk Performer's recording techniques automatically generate scripts with session data that is specific to the recorded test run. Scripts themselves do not contain static session information. So customization of session handling is generally not required for most applications.

- **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.
- **Analyze TrueLog On Error** – TrueLog On Error files provide complete error histories of all errors that occur during tests, making them uniquely well-suited for root-cause analysis. TrueLogs allow you to drill down through real content to analyze error conditions. They display errors in the context of the sessions within which they occur and are closely integrated with test scripts.
- **Visualize and customize database operations** – Database operations such as SQL queries can be customized in the same way that HTML operations are customized. Verifications can also be applied to result-set data.
- **Visualize and customize XML** – TrueLog Explorer includes an intuitive XML viewer that enables close inspection of all XML-based data, including SOAP requests and responses. Customization and visual verification of XML data is also available.
- **Visualize and customize Citrix** – You can record, replay, and customize onscreen events such as mouse moves, mouse clicks, keyboard inputs, and window creations/destructions for Citrix-enabled applications. Screen synchronization functions are also available for verification purposes.
- **Visualize and customize terminal emulation** – You can record, replay, and customize test scripts for terminal emulation protocols `VT100+`, `IBM 3270`, and `IBM 5250`. Test scripts can be customized with value parsing, status-value parsing, value verification, and status-value verification.
- **Analyze TCP/IP and UDP test results** – Record and replay TrueLogs for applications that use UDP and TCP/IP based protocols, such as POP3, SMTP, and custom protocols, can be compared to identify session-dependent information and analyze errors.
- **Analyze GUI-level test results** – Record and replay TrueLogs for the testing of GUI-level (fat client) applications in remote desktop sessions.

Tour of the UI

Take some time to acquaint yourself with the TrueLog Explorer interface to make your test-script customization and verification tasks easier to organize and perform.

The main sections of the TrueLog Explorer interface are highlighted below.



Main Menu

The main menu provides access to TrueLog Explorer results-analysis and script-customization features.

Note: Depending on the protocol type of the open TrueLog, certain menu items may not be accessible.

Toolbar

The toolbar provides access to frequently used features.

Note: Depending on the protocol type of the open TrueLog, certain toolbar buttons may not be accessible.

Workflow Bar

The workflow bar facilitates the primary actions related to analyzing test-run results and customizing scripts. The workflow bar reflects TrueLog Explorer's built-in load-testing methodology by supporting its five primary tasks:

- Analyzing test runs
- Customizing session handling in test scripts
- Adding verification functions to scripts
- Analyzing TrueLog On Error

API Node Tree Menu

The menu tree, located on the left side of the interface, allows you to expand/collapse each TrueLog API node (via double-click). Click a node to display its contents on the Content pane and its history details on the **Information** page.

To view details about a Web page that is included in a TrueLog, select the page's URL or link description in the menu tree. The content is then displayed on the Content pane and its properties are displayed on the **Information** page.

Content Pane

TrueLog Explorer provides multiple views of all received data. Depending on the type of application under test (Web, XML, database, TCP/IP, UDP, Oracle Forms, SAPGUI, Citrix, or other), the Content pane offers different viewing options:

Rendered *(HTML and XML only)*

The **Rendered** page displays the visually rendered response from the server for the selected API node in the menu tree. HTML responses are rendered by an Internet Explorer control running behind the **Rendered** page. XML responses are displayed by an XML control that displays XML data in a menu tree format. Responses that are neither HTML or XML are not rendered and must be viewed using either the **Source** page or the **in-data** page. You can right-click elements and attributes in the XML control to expand the menu tree by one or more levels.

SQL Command *(database applications only)*

The **SQL Command** page displays SQL statements associated with the selected database call, including input and result data. It is the same as the **Rendered** page for other protocols.

Source *(HTML, XML, TCP/IP and UDP only)*

Presents raw source data. The **Source** page displays the HTML code that is used to generate Web content. It contains the same information that the **in-data** page includes. Using the **Source** and **out-data** pages, you can view the in-data and out-data of requests at the same time.

Source Differences *(HTTP-based applications only)*

The **Source Differences** page lists the in-data (source) differences between selected nodes. This page is helpful in finding dynamic information in server responses. It is the first place to look for session information.

Note: The **Source Differences** page is only available in *difference mode.*

Post Data *(HTML and XML only)*

The **Post Data** page displays out-data (data sent by an application to a server) in a rendered format. With XML, an XML menu tree representation is displayed. With HTML, posted data (either the POST data of a HTTP-POST command or the query string data of a HTTP-GET command) are displayed within the HTML page that includes the HTML form that corresponds to the posted data. Only context-full HTML form data submissions (HTML form data submitted with `WebPageSubmit`, not submissions via `WebPageForm`) are displayed on the **Post Data** page. Elements are displayed in magenta. Attributes are displayed in cyan.

To search for content in rendered view, right-click in the rendered view and choose **Find**. The global find option is not able to search rendered view.

Note: Rendered and **Post Data** pages can display content from different API nodes simultaneously. When the **Rendered** page is the result of an HTML-FORM

submission of a preceding Web page, the **Post Data** page displays the Web page that contains the submitted values (which are visible via mouse-over).

Form Controls *(Oracle Forms only)*

The **Form Controls** page displays the properties of Oracle Forms controls (state and content). In compare mode, this page compares the states of controls between record and replay.

Screen *(Citrix only)*

The **Screen** page displays rendered application GUIs, window synchronization functions, and user input values.

Window *(Citrix only)*

Displays the window that is referenced by the window parameter of the current function.

Start Request *(SAPGUI only)*

The **Start Request** page displays the state of a SAPGUI application before each server round trip.

End Request *(SAPGUI only)*

The **End Request** page displays the state of a SAPGUI application after each server round trip.

Host Screen *(Terminal-emulation applications only)*

The **Host Screen** page displays the visual rendered response from the server for the selected API node in a terminal view.

Information Pane

The **Information** pane offers data regarding testing scripts and test runs, including BDL scripting, HTTP headers, and timing statistics. Depending on the type of application under test (Web, XML, database, Oracle Forms, SAPGUI, Citrix, TCP/IP, UDP, or other), the **Information** pane offers different views.

Info

The **Info** page displays general information about the open TrueLog file and the selected API node, including:

- Script file name
- Function
- Line number
- Time
- Duration
- Absolute URL
- Completion status
- Additional information, if available

BDL

The **BDL** page displays the BDL script that corresponds to the open TrueLog. The BDL script is automatically positioned to the line of the selected API node.

References *(HTML only)*

This page is active only for those nodes that return HTML data. The **References** page displays a simple DOM (Document Object Model) of the selected HTML. The page includes the following HTML elements:

- Hyperlinks
- Frames
- Embedded documents
- Forms

out-hdr *(HTTP only)*

The **out-hdr** page contains the exact HTTP header that the application (Silk Performer in the case of replay TrueLogs, the browser in the case of record TrueLogs) sends to the server.

in-hdr *(HTTP only)*

The **in-hdr** page contains the exact HTTP header that the server sends to the application.

out-data *(HTTP, Oracle Forms, Citrix, terminal emulation, TCP/IP, and UDP only)*

In the case of HTTP, this page contains data sent with HTTP-POST commands from the application to the server. For TCP/IP and UDP, this page contains data sent with `WebTcpipSend` functions from the application to the server. For Citrix, this page contains data strings. When using the Oracle Forms log levels of `Normal` and `Debug` (this can be set in replay profile settings), the **Out-Data** page contains Oracle Forms messages that are sent for current actions (for example, clicking a button).

You can switch between hexadecimal and textual presentation of data (**Settings** > **Options** > **Display** > **Data Format**). By default TrueLog Explorer selects the best representation of data (for example, binary for images, text for HTML documents).

in-data *(HTTP, Oracle Forms, TCP/IP, terminal emulation, and UDP only)*

The **in-data** page contains data received by the application from the server. The data is presented in raw format as it is received from the server (no rendering for HTML, no menu tree representation for XML).

Statistics *(for Web business transaction - HTTP applications only)*

The **Statistics** page shows timing statistics for Web pages. It includes timing information for each Web page component and communication element. It shows exact response times, including the composition for each individual component, in graphical view. This assists you in pinpointing the root causes of errors and slow page downloads. The **Statistics** page includes the following data for each page component:

- DNS lookup time: The time taken to resolve an IP address from the domain / host name supplied
- Connect time: The time taken for the simulated user to connect to the server
- Time for SSL handshake: The time taken to exchange encryption keys with the server
- Time for sending of data: The time from the first byte of the client request until the last byte of the client request
- Server busy time: Measures the time from the last byte of the client request until the first byte of the server response
- Response receive time: The time from the first byte of the server response until the last byte of the server response, including all docs but not embedded objects
- Cache statistics

Statistics *(for Web browser driven - AJAX applications only)*

The **Statistics** page shows timing statistics for Web pages. It includes timing information for each Web page component and communication element. It shows exact response times, including the composition for each individual component, in graphical view. This assists you in pinpointing the root causes of errors and slow page downloads. The **Statistics** page includes the following data for each page component:

- DNS lookup time: The time taken to resolve an IP address from the domain / host name supplied
- Connect time: The time taken for the simulated user to connect to the server
- Net round trip: The time from the first byte of the client request until the last byte of the server response, including all documents but not embedded objects.
- Cache statistics

Form Data *(HTTP and Oracle Forms only)*

A convenient tool for viewing and working with all customizable controls that are included on Oracle Forms screens or submitted to Web servers.

Controls *(SAPGUI only)*

The **Controls** page is a convenient tool for viewing and working with customizable control values that are included within SAPGUI windows.

Host Screen Info *(Terminal-emulation applications only)*

The **Host Screen Info** page logs additional information for each host screen, including status (a name/value pair). When a field name is selected, the corresponding field in the **Host Screen** window above is selected. General screen information. For example, cursor position and screen dimensions is also included.

Best Practice Use of TrueLog Explorer

Best practice use of TrueLog Explorer involves the following steps.

Most likely you will not perform these steps in a simple one-time-through sequence. These tasks are to be performed iteratively. For example, you may make a customization, check the outcome, then make another customization, and check the outcome of that customization.

Note: To enable the insertion of timers and other script-modification features, your TrueLog must be up-to-date (in sync). If your TrueLog is not up-to-date, execute a Try Script run to synchronize it.

1. **Analyzing a Test** — This process involves examining the outcome of a previous test run, reviewing a Virtual User Summary Report, locating replay errors, and comparing a replay session alongside an original record session. This process allows for the error checking of script customizations.
2. **Customizing Session Handling** — Session-handling errors occur when outdated session data that is embedded in a load-test script is rerun in a subsequent test run. TrueLog Explorer automates the process of identifying such static data and replacing it with dynamic data to facilitate successful test runs. No manual editing of code is required.

   Note: Because the Silk Performer script recording techniques generate context-full scripts that do not contain static session information, session handling customization is generally not required.
3. **Customizing User Data** — To better simulate real-world conditions during load testing, the actions that virtual users take against servers should vary with each simulated transaction. User input data customization allows the specification of data files (for example, lists of names/

addresses, numbers, and products) from which data is to be pulled when the test script simulates such user tasks as form-field data entry. With random functions, randomized data can also be generated for input fields.

4. **Adding Verification Functions** After the completion of customizing how a test script handles session information and virtual user data input, functions can be built into scripts to automatically check if the application under test returns accurate data. Such content verifications confirm whether or not elements, such as graphics and data, are actually received by clients under real-world conditions. When such elements are not received, verifications raise errors.

5. **Extending Customization via Silk Performer** Although TrueLog Explorer automates most BDL scripting automatically, more sophisticated verifications and customizations can be coded into scripts manually via Silk Performer.

6. **Analyzing TrueLog On Error** After a test has been run, TrueLog On Error files provide complete histories of all erroneous transactions that are encountered during testing. TrueLog On Error enables you to drill down through real content to perform root-cause analysis on system and application faults.

Understanding TrueLog

Provides an overview of the history files in which extensive detail regarding data/page requests and server responses are recorded during testing.

TrueLog Overview

Because Silk Performer links sessions with test scripts and captures all data sent between clients and servers, it is uniquely able to incorporate replay results into the customization and verification of scripts.

This TrueLog technology represents a tremendous advantage in error reproduction and root-cause analysis. TrueLogs are history files in which extensive detail regarding data and page requests and server responses are recorded during testing. For Web testing, this detail includes complete page contents. Such information is valuable for identifying the causes of system failures.

TrueLogs allow you to drill down through real content that is captured during error conditions in client-server and Web-based applications. For HTML applications, TrueLog Explorer allows for the analysis and manipulation of TrueLog files by way of an intuitive, rendered HTML interface that mirrors what virtual users see during testing, including session context.

Because TrueLogs are tightly coupled with sessions and test scripts, they enable script customization in real-time. When a script customization is made, its effects can instantly be checked using a trial script run, or *Try Script run*, by reviewing the TrueLog to determine whether the customization worked. If a customization does not work, a TrueLog shows you the system requests and responses in the session that led to the error. This process can be repeated until all customizations operate as intended.

TrueLog and TrueLog On Error files carry the file extension `.xlg`. `XLZ` files are zipped TrueLog files that are handled as unzipped TrueLog files within TrueLog Explorer. Zipped TrueLogs minimize download times. Silk Performer does not use zipped TrueLogs.

Visual Verification Under Load

Web-based applications do not commonly use HTTP error codes to send error information. Instead, they typically send error information within HTML content. In such scenarios, errors are normally detected only when scripts attempt to call links on pages that include error messages rather than their original content. Such attempts typically lead to `link not found` errors. Such messages generally are insufficient for uncovering the reasons why errors occurred and determining how errors can be reproduced in debugging environments. The only way to track down the causes of such elusive errors is to record error histories under load and determine the actions of the virtual users and the server before the error occurred.

TrueLogs provide this information even when several thousand virtual users are simulated and errors occur only occasionally over long periods of time.

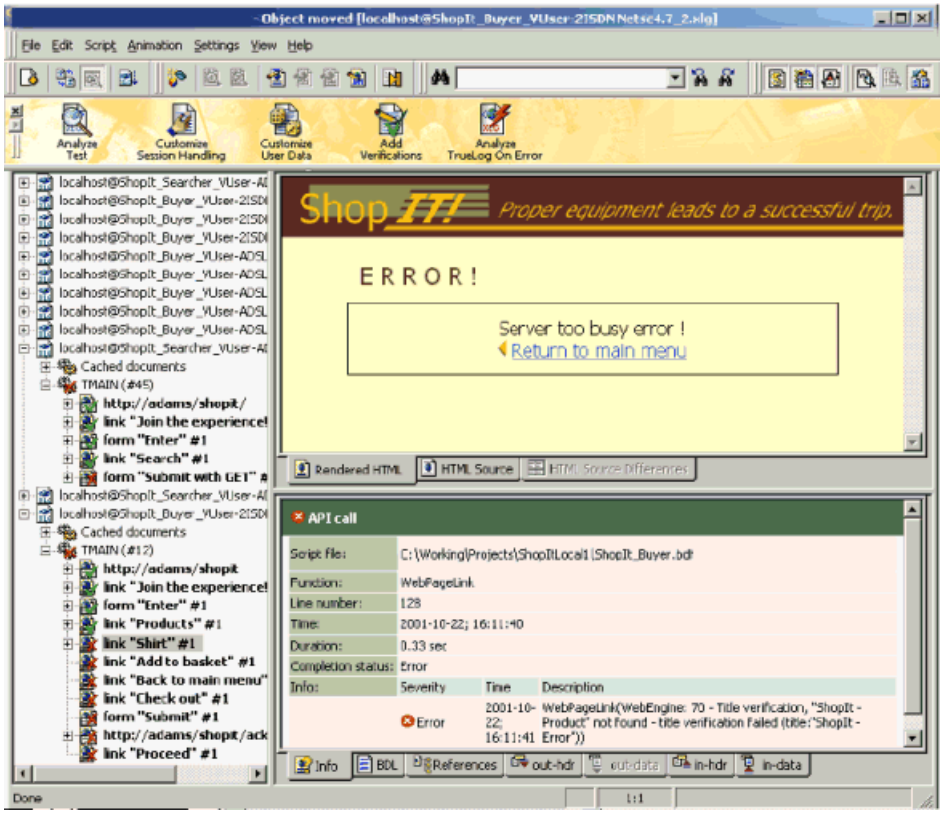
TrueLogs facilitate *visual verification under load*, which reveals elusive application-level errors that usually are encountered only by subsets of users when applications are under heavy load. For most applications, this sort of load is only experienced once an application is deployed. Typical non-HTTP errors for Web applications include incorrect text on Web pages, incorrect value computations, and application-related messages such as `Servlet Error` and `Server Too Busy`.

Request and Response Logs for Root-Cause Analysis

Visual verification under load on its own does not provide much value because it does not indicate the root causes of application-level errors. For this reason TrueLogs record all relevant user interactions and server responses, including those that occur in the moments preceding errors.

A virtual user may run into an application-level error if a `link not found` error is displayed when a returned Web page does not include a certain link. The Web server might not have been able to return the complete Web page, or the application server might not have been able to deliver the page's dynamic content in a timely manner. Using TrueLog Explorer to analyze the TrueLog history that was recorded in the moments preceding such an error allows you to visually inspect the click path that led to the error. Such TrueLog history offers valuable information that can be used to identify the root cause of the error and ultimately fix it.

The following image offers a visual log that reveals the click path of a transaction that led to an error. This TrueLog reveals that a title-verification error resulted from an error message returned by the Web server.



TrueLog On Error Files

Because TrueLog files can become large in size, Silk Performer can be configured to generate TrueLogs only when they encounter errors during load testing. Nothing is recorded while systems run accurately.

Such targeted TrueLog generation is known as *TrueLog On Error* and results in smaller TrueLog files that are focused on error conditions.

When errors are encountered during test runs, TrueLog On Error files reveal the sources of the errors. After a test run, click **Analyze TrueLog On Error** on the workflow bar to access and analyze the corresponding TrueLog file.

Stepping Through TrueLogs

The **Step through TrueLog** dialog box assists you in advancing through recorded TrueLog data at a specified interval.

1. Select a TrueLog in the **TrueLog** menu tree. TrueLog content appears on the **Rendered** and **Information** pages.
2. Choose **Edit** > **Step through TrueLog** .

   Alternative: Click **Step through TrueLog** on the toolbar.

   The **Step through TrueLog** dialog box opens.
3. Specify the interval at which you want to advance through the TrueLog's recorded-data nodes.
4. Click **Find Next** and **Find Previous** to navigate between nodes at the specified interval.

Specifying Protocol-Based TrueLog Types

The TrueLog Explorer feature set varies based on the protocol of the open TrueLog. For example, the **Form Data** tab is available only when a Web TrueLog is open; the **Round trips** option is available on the **Step through TrueLog** dialog box only when an SAPGUI TrueLog is open; and the **Stable Screen** option is available on the **Step through TrueLog** dialog box only when a terminal emulation TrueLog is open

1. Choose **Edit** > **TrueLog Type** .
2. Choose the feature set that you want TrueLog Explorer to load.

   The default protocol is **Web**.

   Note: Oracle Forms is a mixed protocol that often incorporates Web calls in its scripts. You may want to select the Web-based protocol feature set to customize session handling, parse values out of HTML, and run searches on the HTML included in Oracle Forms scripts.

Analyzing Tests

Once a BDL-formatted test script has been recorded and saved using the Silk Performer Recorder, you should test the script by performing a Try Script run. Try Script runs are not actual load tests; they are trial test runs that provide you the opportunity to see if your script requires debugging. It is recommended that you perform a Try Script run each time a customization is added to a script. When a Try Script run is initiated via the **Try Script** button, TrueLog Explorer opens automatically with the TrueLog of the most recent test run open.

Test analysis with TrueLog Explorer involves the following three tasks:

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

Visual Analysis

TrueLog Explorer can visually render the Web content that is displayed by applications under test, showing you exactly what your virtual users see during tests.

Though SQL commands, Oracle Forms controls, TCP/IP operations, and UDP operations do not have high-level interfaces comparable to Web content, TrueLog Explorer compare mode allows the comparison

of recorded database, Oracle Forms, Citrix, TCP/IP, and UDP TrueLogs with replayed database, Oracle Forms, Citrix, TCP/IP, and UDP TrueLogs.

Opening a TrueLog

1. Choose **File** > **Add TrueLogs** or **File** > **Add TrueLog On Error Files** . The **Open** dialog box opens with the specified file type selected in the **Files of Type** list box.
2. Navigate to the directory of the relevant test and select the file to be explored.
3. *Optional:* Check the **Open in Compare View** check box to open the file in Compare view.
4. Click **OK**.

Note: Multiple TrueLogs can be open in TrueLog Explorer simultaneously.

Analyzing a Test Run

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. Select one of the following options:
   - **Show the virtual user summary report**
   - **Find errors**
   - **Compare your test**

Virtual User Summary Reports

Explains how virtual user summary reports summarize Try Script runs with basic descriptions and timing averages.

Virtual User Summary Reports Overview

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

Displaying a Virtual User Summary Report

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 **Show the virtual user summary report** link.

Finding Errors

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**).

Inserting Timer Functions

To insert a new timer session, you must load a replay TrueLog and enable TrueLog Explorer *Explorer* mode.

Note: To enable the insertion of timers and other script-modification features, your TrueLog must be up-to-date (in sync). If your TrueLog is not up-to-date, execute a Try Script run to synchronize it.

Measure functions, or *timers*, can be inserted into BDL scripts to measure the amount of time that transpires from the execution of one transaction to the execution of the next transaction. The results of measure functions are displayed along with summary reports for virtual users.

Note: You can insert a new timer session between the `MeasureStart` and `MeasureStop` functions of any preexisting timer.

1. Right-click an API node in the menu tree where the timer is to begin and choose **Start New Timer**. The **Start Timer** dialog box opens.
2. Enter a name for the timer and click **OK**. A new `MeasureStart` function is inserted into the BDL script at that node.
3. Right-click the API node where the timer is to end and choose **Stop Timer**. The **Stop Timer** dialog box opens.
4. Select the timer that you want to stop and click **OK**. A new `MeasureStop` function is inserted into the BDL script at that node.

Note: When inserting multiple timers into a single script, or if you are uncertain if you have stopped all inserted timers, use the `New Timer Session` function to stop all previous timers before inserting timers. `New Timer Session` inserts a `MeasureStop` function for all unresolved timers before inserting a `MeasureStart` for a new timer session.

Performance Analysis (HTTP)

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:

- DNS lookup time
- Connection time
- SSL-handshake time
- Send-request time
- Server-busy time
- response-receive time
- Cache statistics

Performance Analysis (AJAX)

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 action times and 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:

- DNS lookup time
- Connection time
- Net round trip
- Cache statistics

Viewing an Overview Page

1. Select an API node for which you would like to view statistics.
2. Click the **Statistics** tab. The **Statistics** view opens.

Replay and Record 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.

Replay and Record TrueLogs Overview

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.

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.

For Web testing, TrueLog Explorer provides different views of received data in the Content pane:

- Rendered HTML (as virtual users see it) via the **Rendered** tab
- Native HTML code via the **Source** tab.
- Difference tables that reveal the differences between replay and record TrueLogs via the **Source Differences** tab
- Posted data (pages that include user interface controls such as text entry fields) via the **Post Data** tab

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.

Tip: If your analysis shows invalid session handling, click **Customize Session Handling** on the workflow bar.

Note: Multiple entry points and approaches are available for the comparison of TrueLogs. Replay TrueLogs do not necessarily need to be open in the replay menu tree and record TrueLogs do not necessarily need to be open in the record menu tree. Two TrueLogs can even be compared with one another. The procedure is the same for all TrueLog comparisons.

Comparing Replay and Record TrueLogs

1. Choose **File** > **Add TrueLogs** or **File** > **Add TrueLog On Error Files** . The **Open** dialog box opens with the specified file type selected in the **Files of Type** list box.
2. Navigate to the directory of the relevant test and select the file to be explored.
3. *Optional:* Check the **Open in Compare View** check box to open the file in Compare view.
4. Click **OK**.
5. Click **Analyze Test** on the workflow bar. The **Workflow - Analyze Test** dialog box opens.
6. Click **Compare your test**. The corresponding record TrueLog opens in **Compare** view and the **Step through TrueLog** dialog box displays with the **Whole pages** option selected. With this selected, node-by-node comparison of the TrueLogs can be conducted.
7. Click **Find Next** to step through TrueLog result files one page at a time.

   Experiment with different increments by which to advance through the TrueLogs to visually verify that your script ran as intended (**HTML documents**, **Errors**, **Form submissions**, or **API calls**).

   Note: Sub-panes related to record sessions have red triangles in their upper-left corners, while sub-panes related to replay sessions are marked with green triangles.

Synchronizing Replay and Record 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.

Auto-Syncing Replay and Record TrueLogs

In compare mode you can maintain automatic synchronization between corresponding API nodes in replay and record TrueLogs.

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. Synchronize the TrueLogs by doing one of the following:
   - Click **Keep TrueLogs Synchronized** on the toolbar.
   - Choose **View** > **Keep TrueLogs Synchronized**.

   Now when you click an API node in either TrueLog, TrueLog Explorer automatically selects the API node in the other TrueLog that best correlates with the selected API node.

TrueLog On Error

TrueLog On Error files provide complete histories of the transactions that precede errors uncovered during load tests. They enable you to drill-down through real content to analyze error conditions.

TrueLog On Error files maintain histories of all client requests and server responses. Because they present errors in the context of the sessions within which they occur and are closely integrated with test scripts, TrueLog On Error files are uniquely suited for root-cause analysis of system and application faults.

Because TrueLog On Error is recorded only when errors are encountered, it is much less of a drain on memory and processing than is standard TrueLog.

It is typical after running a test to have multiple TrueLog On Error files open in TrueLog Explorer (one TrueLog for each virtual user who returns an error). With the TrueLog Explorer **Find Errors** feature, errors can be identified chronologically, regardless of which TrueLogs the errors were recorded in. This simplifies the process of analyzing errors. There is no need to manually review all open TrueLogs to find the next error in a sequence.

Analyzing TrueLog On Error

Complete a load test.

1. Click the Silk Performer **Explore Results** button. The **Workflow - Explore Results** dialog box appears.
2. Click **TrueLog Explorer**.

   Note: The **TrueLog Explorer** option is disabled if no TrueLogs are found.

   TrueLog Explorer launches with all of the TrueLog On Error files that were generated for the current test open. The **Find Error** dialog box is displayed.

   Note: If more than 100 TrueLog files are associated with the test, a dialog appears stating that only the first 100 TrueLogs will be opened automatically.
3. Select **All Open TrueLogs** (the default) from the **Search in** list box.

   Search sequentially for errors in all open TrueLogs.
4. Select a TrueLog in the menu tree
5. Select **selected TrueLog only** from the **Search in** list box.

   Search sequentially for errors in only a selected TrueLog.

   Note: TrueLog On Error files are listed chronologically.
6. Select the type of error you're looking for (*error, warning or informational*) in the **Find what** portion of the dialog.
7. Specify a search strategy (*Start from selected node or Start with first error*) in the **Search strategy** portion of the dialog.
8. Navigate between errors using the **Find Next** and **Find Previous** buttons.

   Tip: When testing Web applications, errors often occur one or two steps before they manifest themselves in page content as non-loading images, error messages, etc.

Closing a TrueLog

Once you have finished analyzing a TrueLog, you may want to close it to clean up the menu tree.

1. In the TrueLog menu tree, select the TrueLog that you want to close.
2. Choose **File** > **Remove selected TrueLog** .

   Alternative: To close all open TrueLogs, choose **File** > **Remove all TrueLogs** .

Working With Silk Performer

It is assumed that you are already familiar with Silk Performer functionality. Refer to Silk Performer Help for full details regarding its use and interaction with TrueLog Explorer.

Exploring TrueLogs from Silk Performer

TrueLog Explorer opens automatically with the most recently generated TrueLog when you initiate a Try Script run by clicking **Try Script** on Silk Performer's workflow bar. Additionally, TrueLog Explorer can be opened from Silk Performer by any of the following methods:

- The replay TrueLog of the most recent Try Script run can be opened at the corresponding node by right-clicking within the script in Silk Performer's script editor and choosing **Explore Recent Try Script TrueLog**.
- The replay TrueLog of the current run can be opened at the corresponding node by right-clicking a virtual user output in Silk Performer's **Output** view on the **Virtual User** page and choosing **Explore TrueLog**.
- Record TrueLogs can be opened by right-clicking scripts in Silk Performer's project menu tree and choosing **Explore Recorded TrueLog**.
- The replay TrueLog of the most recent Try Script run can be opened by right-clicking a script in Silk Performer's project menu tree and choosing **Explore Recent Try Script TrueLog**.
- Replay TrueLogs can be opened by choosing **Results** > **Explore TrueLog** .
- The replay TrueLog of the current run can be opened by right-clicking virtual users in Silk Performer's **Monitor/Virtual User** view and choosing **Explore TrueLog**.

Note: Click **Explore Results** on the workflow bar to analyze TrueLog On Error.

Enabling TrueLog Via Silk Performer

TrueLog Explorer relies on Silk Performer for the writing of TrueLog and TrueLog On Error files. With Try Script runs, which are the most common scenario for use of TrueLog files, TrueLog files are automatically enabled. No configuration is necessary.

Because normal TrueLog generation can produce large amounts of data and degrade performance, consider TrueLog On Error, which only records TrueLog information when transactions fail.

The following table identifies the options for enabling TrueLog and TrueLog On Error:

Options	Steps
Silk Performer **Results** toolbar	Click **Generate TrueLog Files** on the Silk Performer **Results** toolbar to record all TrueLog activity, or click **Generate TrueLog On Error Files** to generate TrueLog only when failed transactions occur.
Silk Performer menu	Choose **Settings** > **Active Profile** > **Results** > **TrueLog** . Check the **TrueLog files (.xlg)** check box to record all activity, or check the **TrueLog On Error files (.xlg)** check box to generate TrueLog only when failed transactions occur. Click **OK**.
Silk Performer's **Workload Configuration** page	To enable TrueLog On Error for your tests, check the check box in the **Settings** section on Silk Performer's **Workload Configuration** page.

Try Script Runs

Try Script runs are Silk Performer test runs that are used to evaluate the readiness of test scripts. In Try Script runs, original recorded sessions are compared against replay sessions to identify session information that may require parsing or to test verification functions.

Executing a Try Script Run from TrueLog Explorer

Choose **Script** > **Do a Try Script Run** . Silk Performer then opens and runs the specified test script.

Executing a Try Script Run from Silk Performer

1. On the Silk Performer workflow bar, click **Try Script**. The **Try Script** dialog box opens.
2. Click **Run**.

Note: To open TrueLog Explorer during the Try Script run, select the **Animated Run with TrueLog Explorer** check box on the **Try Script** dialog box.

TrueLog Explorer for Silk Test

An alternative version of TrueLog Explorer ships with Silk Test, Silk's functional-testing tool. The version of TrueLog Explorer that accompanies Silk Test is set to *Viewer mode,* meaning that Silk Test TrueLogs support only the examination of results within test cases. They do not offer script customization, which is available for most Silk Performer-based TrueLogs.

Silk Performer utilizes Silk Test to execute GUI-level (fat client) tests in remote-desktop sessions. For full details, see the Silk Performer Help.

For information about Silk Test's interaction with TrueLog Explorer, refer to the TrueLog Explorer Help that ships with Silk Test.

Perspectives

TrueLog Explorer offers two viewing modes, or *perspectives,* to accommodate various TrueLog types. The default, most commonly used view perspective for Silk Performer users is the *Explorer* perspective. Explorer perspective offers the full range of script customization features offered by TrueLog Explorer. Explorer perspective enables all functionality offered by the TrueLog Explorer workflow bar. By default, attended Silk Performer tests always produce Explorer-perspective TrueLogs.

Viewer perspective enables only a subset of the context-menu commands across all TrueLog Explorer views and disables the workflow bar. The intent of this perspective is to offer a simplified view for users who are not customizing their scripts, and to support TrueLog types for which script customization is unavailable. By default, unattended Silk Performer tests and Silk Test tests always produce Viewer-perspective TrueLogs.

Switching to Viewer Perspective

Toggling between Explorer and Viewer perspectives is enabled for TrueLog types that offer the Explorer perspective. For TrueLog types that do not support the Explorer perspective, Viewer perspective is the only available mode.

Choose one of the following options:

- Click **Enable Viewer perspective** (the eye icon on the toolbar).
- Choose **View** > **Perspective** > **Viewer** .

Switching to Explorer Perspective

Toggling between Explorer and Viewer perspectives is enabled for TrueLog types that offer the Explorer perspective. For TrueLog types that do not support the Explorer perspective, Viewer perspective is the only available mode.

Choose one of the following options:

- Click **Enable Explorer perspective** (the magnifying glass icon on the toolbar).
- Choose **View** > **Perspective** > **Explorer** .

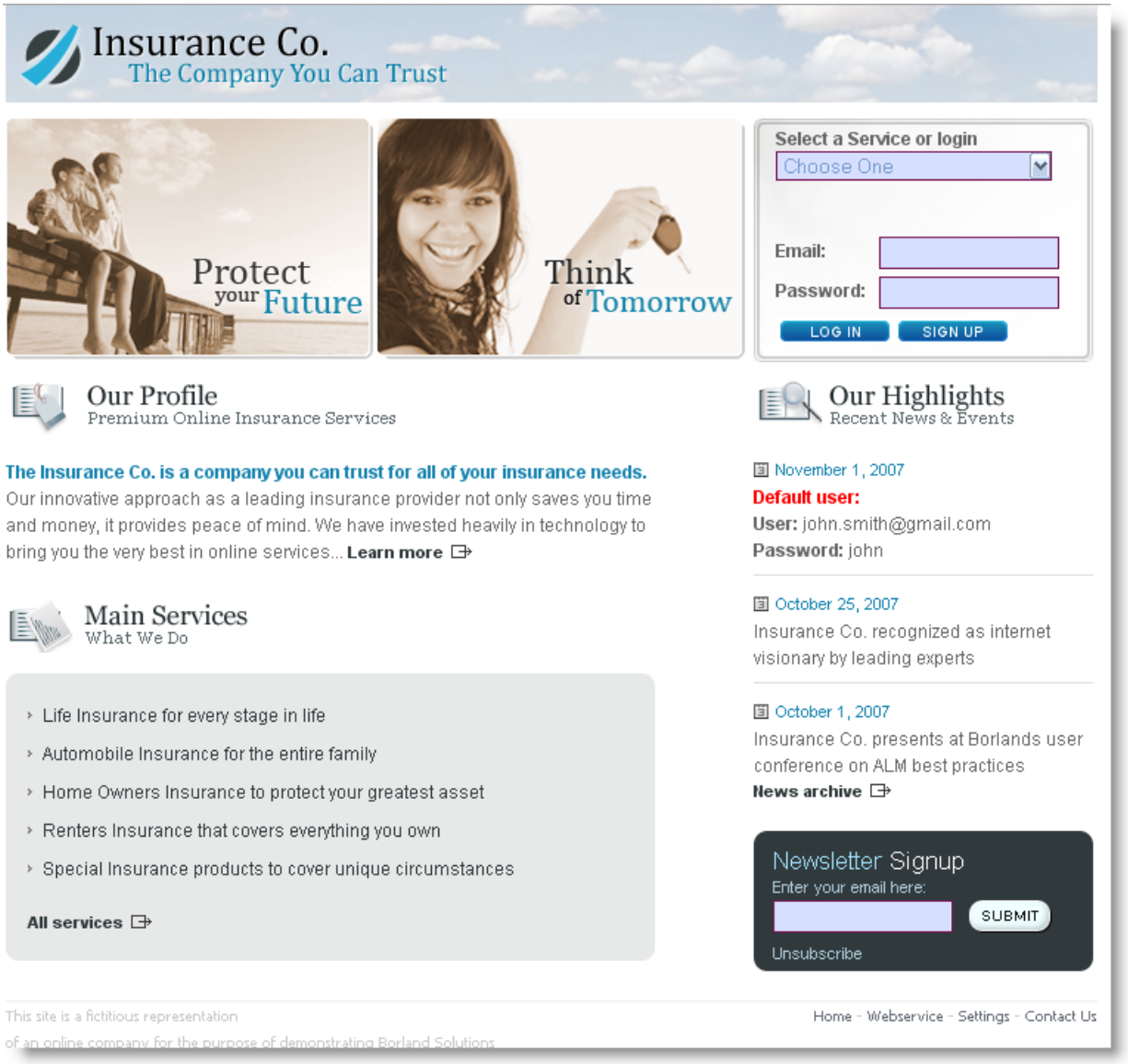
Sample Applications

This Help references three sample applications: a Web 2.0 application, a classic pure HTML Web application, and a database application. Use these applications to acquaint yourself with Silk Performer's and TrueLog Explorer's functionality before you begin working with sensitive data.

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/*.



Sample Classic Web Application

ShopIt simulates a simple, pure HTML e-commerce Web site with a catalog of camping merchandise that is available for simulated online purchase. Use this application to experiment with TrueLog Explorer's pure HTML Web-application capabilities.

ShopIt Overview

ShopIt simulates a simple e-commerce Web site with a catalog of camping merchandise that is available for simulated online purchase. Use this application to experiment with TrueLog Explorer's Web-application capabilities.

ShopIt is designed to generate errors, including session errors and missing Web links that are the result of out-of-stock merchandise. The session errors are the result of session information that is embedded in JavaScript and is not detectable by Silk Performer's context management. The test script that includes this hard-coded session data must be customized before the script can run error-free.

Note: See the `ShopItV60.exe` readme, which is available in the setup, for the latest information about ShopIt.

Software Requirements for ShopIt

A Windows operating system that includes IIS 4 or IIS 5 with Active Server Pages and Microsoft Internet Explorer (version 5.0 or later) is required.

Installing ShopIt V 6.0

The Silk Performer sample web application is ShopIt V 6.0. ShopIt V 6.0 simulates a simple e-commerce website 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 links (due to merchandise being out of stock) and session errors.

Before you install ShopIt V 6.0, refer to the *Release Notes* to ensure that your system supports the use of ShopIt V 6.0.

ShopIt V 6.0 setup is available from the following locations:

- You can start it from the Silk Performer installation DVD: `\Extras\ShopItV60.exe`
- You can download it from the *product updates site*.

1. Double-click the file `ShopItV60.exe`

   Note: IIS (Internet Information Server) must be installed on the computer. For IIS 7, also install Role Services ASP and ISAPI Extensions.

   InstallShield prepares the installation and then the **Welcome** page opens.
2. Click **Next**. The **Choose Destination Location** page opens.
3. To change the default installation directory, click **Browse** to open the **Choose Folder** dialog box.

   The default installation destination is displayed in the *Destination Folder* section.

   Specify the folder to which you want to install ShopIt V 6.0, then click **OK** to return to the previous dialog box.
4. Click **Next** to continue the installation process.

   Enter the name of the virtual directory for the web application in the entry field. This is the name of the directory that will be created on the web server. Click **Next** to continue.

   The **Specify Virtual Directory** dialog box opens.
5. Setup installs the files and configures IIS to run the ShopIt V 6.0 web application. The **Installation Complete** dialog box opens.
6. Click **Finish** on the **Installation Complete** dialog box. The ShopIt V 6.0 web application is now ready for use. You can access ShopIt V 6.0 with a browser of your choice by entering the following URL:

   ```
   http://<computer name>/<virtual directory name>/
   ```

   Example:

   If the name of your computer is `JohnSmith` and you have not modified the default value `ShopItV60` for the virtual directory, the URL is:

   ```
   http://JohnSmith/ShopItV60/
   ```

   Or, if you access ShopIt V 6.0 from the computer on which you installed it, the following URL also works:

   ```
   http://localhost/ShopItV60/
   ```
7. For IIS 7: Add the virtual directory to IIS manually.

   - Alias: ShopItV60
   - Physical path: Install directory of ShopIt.

Sample Database Application - Customer OCI

Customer OCI, the sample database application, simulates a simple application that allows you to add, edit, and delete customer contact information.

Customer OCI Overview

Customer OCI, the sample database application, simulates a simple application that allows you to add, edit, and delete customer contact information. Use this application to record sample database sessions that include the insertion of new customer records and database queries. You can use the resulting TrueLogs to acquaint yourself with TrueLog Explorer database functionality.

Two versions of Customer OCI are provided:

- Person PB V6 - OCI7
- Person PB V7 - OCI8

Both versions are identical except they use different versions of PowerBuilder (versions 6 and 7) and different versions of OCI (versions 7 and 8).

Note: The sample TrueLogs were created using Person PB V7 - OCI8.

Software requirements for Customer OCI

To run Customer OCI, you must have:

- Oracle8 or Oracle9 client installation
- Access to an Oracle database

Accessing Customer OCI

Use Customer OCI, the sample database application, to record sample database sessions that include the insertion of new customer records and database queries. Customer OCI is part of the Silk Performer installation.

Determine which version you want to access and then perform one of the following steps:

- To access PersonPB V6 - OCI7, choose **Start** > **Programs** > **Silk** > **Silk Performer 16.0** > **Sample Applications** > **Database Samples** > **PersonPB V6 - OCI7 sample application** .
- To access PersonPB V7 - OCI8, choose **Start** > **Programs** > **Silk** > **Silk Performer 16.0** > **Sample Applications** > **Database Samples** > **PersonPB V7 - OCI8 sample application** .

Customizing Session Handling

Illustrates how to parse out hard-coded session data that sometimes appears in record TrueLogs. Session-handling customization enables you to maintain state information during testing.

Session-Handling Overview

Session-handling customization is the process of manipulating server responses in such a way that application state information is preserved during load testing.

In their responses to clients, servers often generate information at runtime that is used to identify future client requests as coming from the same computer during the same user session. Servers may send out unique strings, commonly known as session IDs. If not updated in test scripts, such session information can create problems in subsequent test runs.

When replayed test runs are compared to originally recorded test runs and outdated session information is discovered, that information must be replaced with dynamic variables in future test runs. Otherwise test scripts pass along invalid session IDs or other session information.

TrueLog Explorer identifies differences that are relevant for customization in yellow and non-relevant differences that do not require customization in blue.

Note: Because Silk Performer recording techniques generate context-full scripts that do not contain static session information, session handling customization is generally not required for most applications. So if you do not detect any problems when you analyze your test you can skip session-handling customization and proceed with user data customization.

Note: The examples presented here relate specifically to Web applications, however, the same principles can be applied to database and XML-based applications. Though Oracle Forms, SAPGUI, Citrix, terminal emulation, TCP/IP and UDP record/replay TrueLogs can be compared using TrueLog Explorer, session-handling customization is not available for those application types.

Determining When to Customize Session Handling

When a `WebPageUrl` call in a script uses a URL that contains a session ID as part of the query string, that same hard-coded static session ID will be sent to the server when the script is replayed. The session ID, however, will not correctly identify the replay session. It will only identify the earlier recorded session, causing the script replay to generate an error.

Without replacing static session IDs in scripts with dynamic values generated at runtime, Web applications are likely to generate errors, such as `Your session has expired. Please return to the login screen.`

Because of Silk Performer session-handling methods, session customization is not required for most Web applications. In the rare instances where manual session customization is required, TrueLog Explorer facilitates the process.

Silk Performer uses two session-ID handling methods that reduce the need for manual handling of hard-coded session IDs:

Cookie Management

Silk Performer automatically handles dynamic session ID values for servers that use cookies to exchange session information. Because Silk Performer accurately emulates browser cookie management, it can send cookies to servers in the same way that browsers do, and thereby eliminate the need for manual interaction to maintain state.

Page-level Web API

Using page-level API for recording (the default setting) delivers scripts that generate context-full Web API function calls, such as `WebPageLink` and `WebPageSubmit`. Context-full Web API calls work at the HTML level, not the HTTP level, and therefore don't use URLs as parameters. Manual session customization isn't required for context-full API calls. The Silk Performer page-level API is used when the application type Web business transaction (HTML/HTTP) is selected in the **Outline Project** dialog box.

Note: It is strongly recommended that the Silk Performer page-level API be used rather than the Silk Performer low-level, browser-based API.

When client-side JavaScript is relied upon for the dynamic generation of HTML, the Silk Performer Recorder occasionally loses HTML context and scripts context-less Web API calls. Context-less Web API calls, such as `WebPageUrl` and `WebPageForm`, contain URLs as parameters. In these rare instances scripts may contain hard-coded session IDs that can be found in the URL parameters of Web API calls and in the form fields declared in the `DCLFORM` sections of scripts.

Example

The following context-full script does not require customization.

```
transaction TMain
  begin
    WebPageUrl("http://myHost/ShopIt/"); // first call 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:
                                            //
LGIJALLCGEBMIBIMFKOEJIMM2
                                            // recognized as a
hidden
                                            // form field, the
value is
                                            // taken from the
actual
                                            // HTML form field.
    "name"            := "Tester",          // changed
    "New Name Button" := "" <USE_HTML_VAL>; // unchanged value:
"Enter"
```

Example

Following is a script with context-free functions (static session data that needs to be customized is included in the `DCLFORM` section):

```
transaction TMain
begin
  WebPageUrl("http://myHost/ShopIt/"); // first call always
context-less
  WebPageUrl("http://myHost/ShopIt/main.asp", NULL,
SHOPIT_MAIN_ASP001);
  WebPageForm("http://myHost/ShopIt/main.asp",
SHOPIT_MAIN_ASP002);
  WebPageUrl("http://myHost/ShopIt/products.asp");
end TMain;

dclform
  SHOPIT_MAIN_ASP001:
    "from"     := "welcome";

  SHOPIT_MAIN_ASP002:
    "SessionID"       := "LGIJALLCGEBMIBIMFKOEJIMM2",
    "name"            := "Tester",
    "New Name Button" := "Enter";
```

Customizing Session Handling

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.

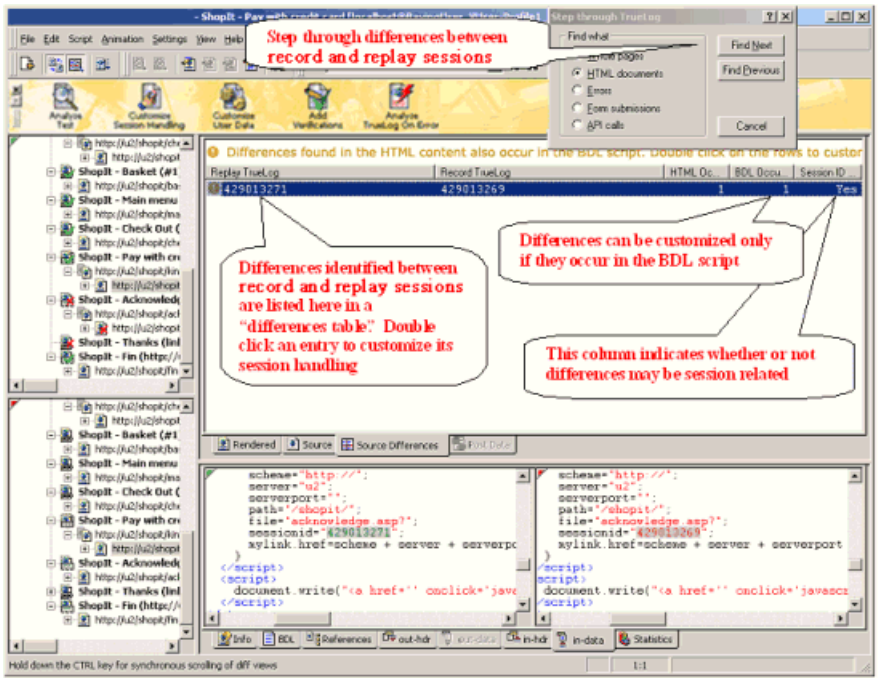
1. Select a TrueLog in the menu tree.
2. Click **Customize Session Handling** on the workflow bar. The **Workflow - Customize Session Handling** dialog box opens.
3. Click the **Find differences** link to view a differences table on the **Source Differences** page.

   This step reveals instances in which the server responded with different (or dynamic) information during the replay session compared to of the record session. Such static information may need to be replaced with a variable.

   Note: When the corresponding record TrueLog is not already open, a dialog box opens asking if you want to have the corresponding record TrueLog opened.

   a) Click **OK**.
4. Step through the HTML server responses using the **Step Through TrueLog** dialog box.

   Recorded responses are displayed alongside the corresponding replayed responses. Only those differences that indicate that static information is included in the test script and being sent back to the server need to be parsed. For example, a difference between replay and record sessions might be due to an e-commerce site running out of stocked merchandise. Such a difference would not be appropriate for script customization because it is not session related.



   Tip: The column headers on the **Source Differences** page offer helpful mouse-over tooltips that describe the elements contained in each column.
5. Double-click one of the errors listed on **Source Differences** page. The **Insert parsing function** dialog box opens with the boundary values already inserted. There is no need to locate and enter these values manually.
6. *Optional:* Click **Create Recording Rule** to create a recording rule based on the values of the parsing function. This enables you to record the application in the future without having to care about session customization again.
7. Back on the **Insert parsing function** dialog box, click **OK**.
8. Click **Customize Session Handling** on the workflow bar once the script has been successfully modified.
9. Click **Try Script Run** to see if the script runs correctly now that session handling has been modified. A new Try Script run is initiated.

10. Analyze the results of the subsequent test run to determine whether or not session handling customization was successful.

Self-Learning Recorder

The Silk Performer self-learning Recorder is adept at supporting applications that rely heavily on client-side scripting. Equipped with advanced context management techniques, the recorder is able to generate Web scripts that maintain page context even when page requests are executed via JavaScript functions.

Maintaining page context means that Web pages are requested in the context of the links or submit forms that call them. Such context-full Web page request functions reference Web pages via their links or form names rather than simply by their URLs. To do this, Silk Performer parses through Web-server traffic for links for form submissions to identify the pages that called the content.

Parsing Functions

Parsing functions can be used to maintain state information during load testing and to simulate real-world user-data input. Parsing functions (which are stored in BDL variables) read session-related values in server responses. They then use parsed values as parameterized data that is sent back to servers as part of a query string post or URL data.

Note: Parsing functions are different from verification functions, which only check for the presence of specified input values.

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.

HTML Content Parsing Functions

HTML content parsing functions can be applied to rendered, visible HTML content. They are applied in TrueLog Explorer **Rendered** view. HTML parsing functions allow for the parsing of HTML content that can be viewed in Web browsers.

The following HTML content parsing functions are available:

- `WebParseHtmlBound` and `WebParseHtmlBoundEx`
- `WebParseHtmlBoundArray` (not supported by TrueLog Explorer)
- `WebParseHtmlTitle`
- `WebParseTable`
- `WebParseResponseTag` (not supported by TrueLog Explorer)
- `WebParseResponseTagContent` (not supported by TrueLog Explorer)

Response Data Parsing Functions

Data parsing functions can be applied to response data returned by servers. In cases where HTML documents are returned from servers, this includes the complete source code of documents. These parsing functions are applied on the TrueLog Explorer **Source** page.

The following data parsing functions are available:

- `WebParseDataBound` and `WebParseDataBoundEx` (these functions replace the obsolete `WebParseResponseData` functions)
- `WebParseDataBoundArray` (not supported by TrueLog Explorer)
- `WebParseResponseHeader` (not supported by TrueLog Explorer)

Note: Visual parsing with TrueLog Explorer is not currently available for TCP/IP and UDP-based applications. However, BDL functions that allow you to parse response data from `WebTcpipRecv` functions are available.

Session Handling for Web Applications

Session IDs are sent to clients in a number of ways. Most often they are included in cookies, hyperlink URLs, URLs of embedded objects, and HTML form fields. Session IDs are likewise returned to servers within cookies, URLs, and HTTP post data. See the examples below:

Example: Session information included in a cookie:

Information sent to the client:

```
Set-Cookie: SessionID=LGIJALLCGEBMIBIMFKOEJIMM; path=/
```

Information returned to the server:

```
Cookie: SessionID=LGIJALLCGEBMIBIMFKOEJIMM
```

Example: Session information included in a URL:

Information sent to the client:

```
<html>
  …
<a href="/ShopIt/acknowledge.asp?
SessionID=LGIJALLCGEBMIBIMFKOEJIMM" >
  Enter Shop
</a>
…
</html>
```

Information returned to the server:

```
GET /ShopIt/acknowledge.asp? SessionID =
LGIJALLCGEBMIBIMFKOEJIMM HTTP/1.1
```

Example: Session information hidden in a form field:

Information sent to the client:

```
<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>
```

Information returned to the server:

```
POST /ShopIt/kindofpayment.asp HTTP/1.1
…
SessionId=LGIJALLCGEBMIBIMFKOEJIMM&name=Jack&paymentButton=Submi
t
```

Session-Handling Customization Process

Session-handling customization typically follows a standard process. The high-level steps outlined below should be followed for each session ID:

1. Determine whether or not a session ID needs to be customized.
2. Search for the first response (Web API call in a script) in which the session ID is sent from the server to the client.
3. Parse the session ID in the response into a variable.
4. Replace all occurrences of the hard-coded session ID in the script with the variable.

Session ID Identification

Because session IDs differ between recorded and replayed sessions, they are easily identified by comparing a record TrueLog to a replay TrueLog.

Session IDs can be found easily with TrueLog Explorer. Click **Customize Session Handling** on the workflow bar to automatically open the recorded TrueLog file that is associated with the open replay TrueLog. By reading through record and replay TrueLogs line by line, the differences in HTML responses between record and replay scripts can easily be identified

Note: Manual line-by-line reading is not required because the **Source Differences** page automatically lists the differences for you.

Session IDs should be customized when differing phrases in server response data from recorded sessions can also be found in the URLs of Web API calls or in form fields included in the `DCLFORM` sections of scripts.

Not all differences between returned data from record sessions and returned data from replay sessions are relevant for session customization. Only when the differences between record and replay sessions are embedded in scripts should session customization be considered. Such instances are indicated by a yellow warning message on the **Source Differences** page and by the column **BDL Occurrences** (when it has a value greater than `0`).

Differences that most likely require customization are indicated by orange exclamation points on the **Source Differences** page. Differences that most likely do not require customization are indicated with blue question marks (replacing the occurrences of such strings in HTML is not recommended; parsing functions may be inserted for them, but replacements will not be made). Blue exclamation points are informational comments only.

Searching for the First response (Web API)

Session-handling customization requires that you search for the first response (Web API call in a script) in which a session ID is sent from a server to a client.

1. Select a replay TrueLog from the menu tree.
2. Click **Compare mode** on the toolbar.
3. Click **Diff Mode** on the toolbar. The **Source Differences** page lists any differences it identifies between the TrueLogs.
4. Right-click a listing on the **Source Differences** page and choose either **Go to First Occurrence in BDL script** or **Go to First Occurrence in HTML**.

Parsing and Replacing Session IDs

Once the session IDs and Web API calls of BDL scripts that return session IDs have been identified, the session IDs must be parsed from the responses to the Web API calls. All occurrences of hard-coded session IDs must be replaced with variables. TrueLog Explorer supports this task so that manual customization of scripts is not required. TrueLog Explorer generates appropriate parsing and replacement statements automatically.

Parsing and Replacing Session IDs

1. Select a replay TrueLog from the menu tree.
2. Click **Customize Session Handling**. The **Workflow - Customize Session Handling** dialog box opens.
3. Click **Find Differences**. A dialog box displays, asking you if you want TrueLog Explorer to locate and display the associated record TrueLog.
4. Click **Yes**. The associated record TrueLog opens in **Compare View** and the **Step through TrueLog** dialog box opens.

   Note: The **Source Differences** page identifies differences between record and replay data that may require customization. Use the **Step through TrueLog** dialog box to step through `Whole pages` of content and advance between API nodes until you find a difference that requires customization.
5. Double-click a listing on the **Source Differences** page to open the **Insert Parsing Function** dialog box.
6. Right-click a listing and choose **Customize Session Handling**. The **Insert Parsing Function** dialog box opens.
7. The **Insert Parsing Function** dialog box offers parameters by which the parsing function can be adjusted. Though the default settings will likely be correct, you can adjust the following :
   a) **Case sensitivity** - Select this option if you want the parsing function to be case sensitive.
   b) **Variable name** - Enter the name of the variable that should receive the result of the parsing function.
   c) **Left boundary** - This field displays the left boundary of the specified text (the text that is located before the specified text).
   d) **Right boundary** - This field displays the right boundary of the specified text (the text that follows the specified text).
   e) **Insert informational statement into script** - Select **Print statement** to insert an informational print statement into the script after the Web page call. This will write the result of the parsing function to the Silk Performer **Virtual User Output** window.
   f) 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 the time measurements of load tests, these files should only be used for debugging purposes and should not be generated for full load tests.
   g) **Replace with variable** - Checking this check box replaces all occurrences of the parsed text with the specified variable.

8. *Optional:* Click **Create Recording Rule** if you want to create a recording rule based on the values you have specified. Refer to the Silk Performer Help for detailed information on creating recording rules.
9. Click **OK**. Parsing and replacement statements required for customizing the handling of the selected session ID are generated.
10. Run a Try Script to verify the customization and ensure that the session handling issue has been resolved. API nodes marked with green check marks indicate that no session errors were encountered during the Try Script run. Red X marks indicate that script replay errors are still present.

Parsing and Replacement Statements - Example

The sample code below shows an example of the parsing and replacement statements that are required for the customization of session handling.

Note: The original session ID has been commented out, but it is still visible in the script.

The `WebParseDataBoundEx` function generated by TrueLog Explorer parses the value found between the boundaries `name=\"` and `\"` in the HTML document returned from the subsequent `WebPageLink` call. It then stores the actual value in the *sSessionInfo1* variable. The parsing function must be specified before the Web API call from which the response data is to be parsed. To facilitate debugging, TrueLog Explorer inserts a `Print` statement that writes the parsed value to the Silk Performer **Virtual User Output** window. The session ID variable is assigned to the *sFormSid1* form field variable, which is used directly in the `DCLFORM` script section in place of the constant session ID value (`858891446`).

Example

```
transaction TMain
begin
  ...
  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://lab3/ShopItV60/kindofpayment.asp",
    "kindofpayment.asp", SHOPITV60_KINDOFPAYMENT_ASP004);
  ...
end TMain;

dclform
  ...
  SHOPITV60_KINDOFPAYMENT_ASP004:
    ...
//    "sid"                       := "858891446";
    "sid"                      := sFormSid1;
  ...
```

Manual Selection of Differences

Selecting differences manually on the **Source** or **In-Data** pages rather than relying on the **Source Differences** page allows for more control over how phrases are parsed out.

For example, using the **Source Differences** page when a session ID is embedded in multiple places in an HTML page, only the first occurrence of the phrase is parsed out. This is true even if the boundaries used for parsing are not adequate.

Manually selecting an occurrence that you want to parse out allows you to select more accurate boundaries.

Selecting Differences Manually

1. Select a replay TrueLog from the menu tree.
2. Click **Compare mode** on the toolbar.
3. Identify a difference between the TrueLogs either on the **Source** or **In-Data** page.
4. Right-click the different phrase in the record TrueLog. If the phrase selected is static data contained in the BDL script, a context menu displays.
5. Choose **Customize Session Handling**. The **Insert Parsing Function** dialog box opens.

You must now configure the parsing function as required.

Parsing Functions in Scripts

Parsing functions must be inserted into scripts at a point before the Web API calls that initiate the parsing/verification of response data. Multiple parse/verification functions can be specified before each Web API call.

The order of parse/verification functions is not relevant. An exception to this rule includes `WebParseDataBound` and `WebVerifyDataBound` using the flag `WEB_FLAG_SYNCHRON`.

Example

Example of a BDL script utilizing a verification function:

```
WebVerifyHtml("Proper equipment leads to a successful trip",
1, ...);
WebPageLink("ShopIt");
```

Using HTTP Parsing Rules

The HTTP parsing example is designed to give you an overview of Silk Performer recording-rule capabilities.

HTTP parsing rules specify when the recorder is to generate the parsing function `WebParseDataBoundEx()` for dynamically changing values and where it substitutes parsing results. Parsing rules enable the recorder to automatically generate working scripts. It typically eliminates the need for visual session customization with TrueLog Explorer.

Refer to the *Silk Performer Advanced Concepts Book*, chapter *Rule Based Recording*, for detailed information on manually scripting recording rules.

Web Application Parsing Rule - Example

ShopIt, the sample Web application, was deliberately built so that the Silk Performer recorder must script the context-less function `WebPageUrl()` with a form definition that contains a session ID. This is achieved by having JavaScript assemble the URL. Recording ShopIt without recording rules results in a script that has a hard-coded session ID.

In this guided HTTP parsing rule example the context less function `WebPageUrl()` is scripted with a form definition that contains a session ID. Upon 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 by modifying the script as shown below:

Example

```
var
  sFormSid1     : string (100);
  sSessionInfo1  : string (100);

  ...

  WebParseDataBoundEx(sSessionInfo1, STRING_COMPLETE,
    "name=\"", 5, "\"", WEB_FLAG_IGNORE_WHITE_SPACE, 1);
  WebPageLink("Check out", "ShopIt - Check Out"); // Link 3
  Print("sSessionInfo1: " + sSessionInfo1);

  ...

  sFormSid1 := sSessionInfo1;
  WebPageUrl(sParsedUrl,
    "Unnamed page", SHOPITV60_KINDOFPAYMENT_ASP004);

  ...

dclform
  ...
  SHOPITV60_KINDOFPAYMENT_ASP004:
    "choice"                  := "CreditCard",
    "price"                   := "115.8",
//     "sid"                     := "858891471";
    "sid"                     := sFormSid1;
```

Custom Recorder Details

The script runs correctly now that it has been customized. However a problem exists in that each script that is to be recorded in the future will also have be customized.

HTTP parsing rules enable the Silk Performer recorder to continue this type of customization automatically in the future; recorded scripts can be generated automatically without manual interaction.

To do this, research must be done into how each session ID can be parsed. The customization offered by TrueLog Explorer reveals the API calls where each session ID first occurs, and the boundaries that can be used to parse each session ID.

Using TrueLog Explorer, the first occurrence of each session ID can be located in the HTML code.

```
<script LANGUAGE="JavaScript">
  function doProcess(mylink)
  {
    scheme="http://";
    server="lab3";
    serverport="";
    path="/ShopItV60/";
    file="kindofpayment.asp?";
    name="858891471";
      price="115.8";
    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 are good choices for the parsing of the session ID.

Now an initial version of an HTTP parsing rule can be written for the Recorder.

```
<?xml version="1.0" encoding="UTF-8"?>
<RecordingRuleSet>
```

```
  <HttpParsingRule>
    <Name>ShopIt V5.1 Session Id</Name>

    <Search>
      <SearchIn>Body</SearchIn>
      <LB>
        <Str>name=&amp;quot;</Str>
      </LB>
      <RB>
        <Str>&amp;quot;</Str>
      </RB>
    </Search>

    <ScriptGen>
      <VarName>ShopItSessionId</Varname>
    </ScriptGen>

  </HttpParsingRule>

</RecordingRuleSet>
```

Since ShopIt session IDs do not appear in HTTP response headers, it is specified that only response bodies are to be searched (using the `Search\SearchIn` attribute).

It is specified that the session ID can be found by searching for the left boundary (the boundary is specified in the `Search\LB\Str` attribute).

Note: The quotation symbol (") has been encoded in XML using the character sequence `&quot;`.

A single quotation symbol marks the end of the session ID. This is specified with the `Search\RB\Str` attribute. Here again, the quotation symbol has been encoded.

Finally, specifics have been defined regarding how the variable for the parsing result is to be named. The name is specified using the `ScriptGen\VarName` attribute.

This rule file can be saved to the Silk Performer `Include` directory to make it globally available to all projects. The file name can be of any length, but the file extension `.xrl` must be used. Alternatively, if the recording rule is to be used with only one project, the file can be saved to the `Documents` directory of a specific Silk Performer project.

Refer to the *Silk Performer Advanced Concepts Book*, chapter *Rule Based Recording*, for detailed information on manually scripting recording rules.

Adding Verifications

This section provides an overview of creating verification checks for HTML page titles, HTML content, HTML tables, and HTML source code.

Note: Content verifications are currently unavailable for TCP/IP- and UDP-based applications.

Verifications Overview

TrueLog Explorer enables you to create Web content verification checks for HTML page titles, HTML content, HTML tables, and HTML source code. By selecting text that you want to verify either in HTML source code or directly in rendered HTML view, all required verification functions can be generated and automatically inserted into your BDL script.

Visual Data Verification

TrueLog Explorer offers you two approaches to content verification. The easiest approach enables you to create content verification checks via a simple point-and-click interface.

Response Data Verification

Response data verification offers a more powerful, but more complex means of verifying server functionality. As with visual data verification, response data verification is also set up via a point-and-click interface. With visual data verification, you work in rendered HTML view and verify content that the application exposes to the end user. With response data verification, you verify data that are returned by servers that are not visible to the end user, such as HTML code and XML code.

Depending on the functions you select, TrueLog Explorer can generate response-data verification functions that apply to complete response data returned by servers or only to selections within specified boundaries.

Verification Checks

Explains the advantages of including verification checks in test scripts.

When to Use Verification Checks

Application errors often do not result in erroneous HTTP responses. More commonly, applications respond with incorrect data values or error messages that are incorporated into HTML content, such as `Servlet Exception Occurred` or `Server Too Busy` error messages. Because checks of HTTP status code do not uncover this class of error, application errors are often overlooked unless verification functions that check for non-standard HTTP errors are built into test scripts.

When verifications are built into test scripts, tests go from being simple load tests to hybrid load/functionality tests. Such scripts can be utilized without incurring significant performance loss, even in large load test scenarios. This functionality allows you to detect a class of error that other load tools are unable to detect because errors that occur only under load are undetected with standard load test scripts.

TrueLog Explorer offers the following means of enhancing test scripts with verification functionality:

- Enabling the Silk Performer Recorder to automatically generate verification checks during recording.
- Allowing users to apply verification checks visually in **Rendered** view via a point-and-click interface. No editing of BDL code is required. TrueLog Explorer automatically adds verification functions to scripts.
- Directly enhancing scripts by manually inserting coded verification functions.

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**.

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

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.

The load testing script should be ready to run without error when the following tasks have been completed:

- Customize how the application handles session information and user-input data.
- Insert any required verification functions.
- Complete any required manual BDL script editing via Silk Performer.

HTML Verification Functions

HTML verifications check rendered, visible Web content. They verify text-based content that is displayed in Web browsers. The following HTML verification functions can be applied in **Rendered** view:

- `WebVerifyHtmlTitle` – Verifies page title.
- `WebVerifyHtml` – Verifies text-based content.
- `WebVerifyHtmlBound(Ex)` – Verifies content within bounds.
- `WebVerifyTable` – Verifies content within an HTML table.
- `WebVerifyHtmlDigest` – Verifies the digest of an HTML page.

Generating a Title-Verification Function

The `WebVerifyHtmlTitle` function checks the titles (contents of HTML `<title>` tags) of selected HTML pages. No text selection is required to apply title verifications.

1. In the menu tree, select the Web page that you want to verify.
2. Perform one of the following steps:
   - Choose **Script** > **Verify Page Title** .
   - Click **Add Verifications** and then click **Verify page title** on the **Workflow - Add Verifications** dialog box.

   The page title is included in the **constant value** text box, and the **constant value** option button is selected.
3. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.
   b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.
4. From the **Verify that the page title** list box, select one of the following choices:
   - **is equal to**
   - **is different from**
   - **contains**
   - **does not contain**
5. Check the relevant check boxes to make the verification case-sensitive or to apply it as a script-wide rule.
6. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
7. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for title verifications:

```
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");
  ...
```

Generating a Text-Verification Function

The `WebVerifyHtml` function verifies selected text in **Rendered** view. TrueLog Explorer automatically detects the number of occurrences of selected text in HTML documents.

1. In the menu tree, select the Web page that you want to verify.
2. In **Rendered** view, select the appropriate text.
3. Perform one of the following steps:
   - Choose **Script** > **Verify Selected Text** .
   - Right-click the selected text and choose **Verify Selected Text**.
   - Click **Add Verifications** and choose **Verify selected text in HTML page** from the **Workflow - Add Verifications** dialog box.

   The selected text is displayed in the **constant value** text box, and the **constant value** option button is selected.
4. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.
   b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.
5. *Optional*: To make the verification more tolerant, adjust the following settings:

   Note: The settings on this dialog box are automatically set to values that guarantee a successful verification for the current page. Only in cases in which you want to make a verification more tolerant should these settings be changed (for example, by changing "**exactly**" "**2**" times to "**at least**" "**1**" time, or by making a verification case-insensitive).

   a) From the **Occurs** list box, select one of the following choices:
      - **exactly**
      - **at least**
      - **at most**

   b) Type a number in the **time(s) in this page** text box.
6. Check the relevant check boxes to make the verification case-sensitive or to apply it as a script-wide rule. A result variable name is displayed in the **Result variable name** text box.
7. *Optional*: Edit the **Result variable name**.
8. Ensure that the **Require boundary strings** check box is unchecked.
9. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
10. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for HTML text verifications:

```
transaction TMain
begin
  ...
  WebVerifyHtml("Hi Tester! ", 1, WEB_FLAG_IGNORE_WHITE_SPACE
|
    WEB_FLAG_EQUAL | WEB_FLAG_CASE_SENSITIVE, NULL,
    SEVERITY_ERROR, nVerifyHtmlResult1);
  WebPageSubmit("Enter", SHOPIT_MAIN_ASP001, "ShopIt - Main
menu");
  ...
```

Generating a Text-Verification Function Within Boundaries

The `WebVerifyHtmlBound(Ex)` function verifies text selected in **Rendered** view. TrueLog Explorer automatically detects unique boundaries that identify the position of selected text.

Note: With Silk Performer, TrueLog Explorer uses `WebVerifyHtmlBoundEx` instead of `WebVerifyHtmlBound`.

1. In the menu tree, select the Web page that you want to verify.
2. In **Rendered** view, select the appropriate text.
3. Perform one of the following steps:
   - Choose **Script** > **Verify Selected Text** .
   - Right-click the selected text and choose **Verify Selected Text**.
   - Click **Add Verifications** and choose **Verify selected text in HTML page** from the **Workflow - Add Verifications** dialog box.

   The selected text is displayed in the **constant value** text box, and the **constant value** option button is selected.
4. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.
   b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.
5. Check the relevant check boxes to make the verification case-sensitive or to apply it as a script-wide rule. A result variable name is displayed in the **Result variable name** text box.
6. *Optional*: Edit the **Result variable name**.
7. Check the **Require boundary strings** check box. This disables the **occurs** frequency settings.

   Left and right boundaries appear in the **Left boundary** and **Right boundary** text boxes. Boundary strings cannot be edited from this dialog box.
8. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
9. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for HTML text verifications within boundaries:

```
transaction TMain
begin
  ...
  WebVerifyHtmlBound("Name:", "Address", "Tester",
    WEB_FLAG_IGNORE_WHITE_SPACE | WEB_FLAG_CASE_SENSITIVE,
    NULL, SEVERITY_ERROR, bVerifyHtmlBoundSuccess2);
  WebPageSubmit("Submit", SHOPIT_KINDOFPAYMENT_ASP002”);
  ...
```

Generating an HTML Text-Verification Function

The `WebVerifyTable` function verifies text selected within an HTML table cell in **Rendered** view. TrueLog Explorer automatically identifies the HTML table as well as the row and column of the selected cell.

1. In the menu tree, select the Web page that you want to verify.
2. In **Rendered** view, select the relevant text in a table cell.

3. Perform one of the following:

   - Choose **Script** > **Verify Selected Text in HTML Table** .
   - Click **Add Verifications** and select **Verify selected text in HTML table** on the **Workflow - Add Verifications** dialog box.

   The selected row and column are displayed in the **Verify that** list boxes. Specify a different row or column as required.
4. From the list box, select one of the following choices:

   - **is equal to**
   - **is different from**
   - **contains**
   - **does not contain**

   The selected text is displayed in the **value** text box, and the **value** option button is selected.
5. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.
   b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.
6. Specify whether the verification is to be **Case sensitive**.
7. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
8. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for HTML table verifications:

```
transaction TMain
begin
  ...
  WebVerifyTable(7, 3, 1, "115.8",
    WEB_FLAG_IGNORE_WHITE_SPACE |
    WEB_FLAG_CONTAINS | WEB_FLAG_CASE_SENSITIVE,
    NULL, SEVERITY_ERROR, bVerifyTableSuccess1);
  WebPageLink("Add to basket", "ShopIt - Basket");
  ...
```

Generating an HTML Digest Verification Function

The `WebVerifyHtmlDigest` function calculates and verifies a Web page digest against a precalculated digest. A *digest* is a character frequency table that is calculated for certain characters within a rendered HTML page. If an HTML document is contained within a frame set, text is verified only for the frame that contains the selected text, not the entire page.

1. In the menu tree, select the Web page that you want to verify.
2. Choose **Rendered** view.
3. Perform one of the following steps:

   - Choose **Script** > **Verify Page Digest** .
   - Click **Add Verifications** and then click **Verify digest** on the **Workflow - Add Verifications** dialog box.

   A name for the digest is displayed in the **Constant name** text box.
4. *Optional*: Edit the **Constant name** text box.

5. In the **Verify** group box, specify how the verification should be applied:
   - **All characters**
   - **Printable characters**
   - **Alphanumeric characters**
6. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
7. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for HTML digest verifications:

```
const
  HTML_DIGEST_1     :=
"\h01910000000011D063032CCB7D023EFBBD020000000000000000000000000000
00000001
602010601010703010108010503020201"
"\h030102010305020101050104030701030102050504020601070501 02";

transaction TMain
begin
  ...
  WebVerifyHtmlDigest(HTML_DIGEST_1, 79, 0, 0, NULL,
    SEVERITY_ERROR);
  WebPageLink("Check out", "ShopIt - Check Out"); // Link 3
  ...
```

Response-Data Verification Functions

Response-data verification functions verify that correct response data is returned by servers. In the case of Web applications, data verifications include comprehensive source code checks.

Response data verification functions can be applied visually by selecting data, such as code or text, in **Source** view. TrueLog Explorer can generate the following data-verification functions:

- `WebVerifyData` – Verifies selected data.
- `WebVerifyDataBound(Ex)` – Verifies selected data within boundaries.
- `WebVerifyDataDigest` – Verifies a data digest.

Generating an HTML Data-Verification Function

The `WebVerifyData` function verifies selected data, such as code or text, in **Source** view. TrueLog Explorer automatically detects the number of occurrences of selected data in selected HTML documents.

1. In the menu tree, select the Web page that you want to verify.
2. Select the relevant data in **Source** view.
3. Perform one of the following steps:
   - Choose **Script** > **Verify Selected Text** .
   - Right-click the selected text and choose **Verify Selected Text**.
   - Click **Add Verifications** and choose **Verify selected text in HTML page** from the **Workflow - Add Verifications** dialog box.

   The selected text is displayed in the **constant value** text box, and the **constant value** option button is selected.

4. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.
   b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.
5. Specify the frequency by which the selected text is to appear, as follows:
   a) From the **Occurs** list box, select one of the following choices:
      - **exactly**
      - **at least**
      - **at most**
   b) Type a number in the **time(s) in this page** text box.
6. Check the relevant check boxes to make the verification case-sensitive or to apply it as a script-wide rule. A result variable name is displayed in the **Result variable name** text box.
7. *Optional*: Edit the **Result variable name**.
8. Ensure that the **Require boundary strings** check box is unchecked.
9. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
10. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for data verifications:

```
transaction TMain
begin
  ...
  WebVerifyData("main.asp?from=welcome", 1,
    WEB_FLAG_IGNORE_WHITE_SPACE |
    WEB_FLAG_EQUAL | WEB_FLAG_CASE_SENSITIVE, 1,
    SEVERITY_ERROR, nVerifyDataResult1);
  WebPageUrl("http://myHost/shopit/", "ShopIt - Greetings");
  ...
```

Generating a Data-Verification Function Within Boundaries

The `WebVerifyDataBound(Ex)` function verifies selected data, such as code or text, in **Source** view. TrueLog Explorer automatically detects unique boundaries that identify the position of selected data.

Note: TrueLog Explorer uses `WebVerifyDataBoundEx` instead of `WebVerifyDataBound`.

1. In the menu tree, select the Web page that you want to verify.
2. Select the relevant data in **Source** view.
3. Perform one of the following steps:
   - Choose **Script** > **Verify Selected Text** .
   - Right-click the selected text and choose **Verify Selected Text**.
   - Click **Add Verifications** and choose **Verify selected text in HTML page** from the **Workflow - Add Verifications** dialog box.

   The selected text is displayed in the **constant value** text box, and the **constant value** option button is selected.
4. *Optional*: To verify against an existing parameter or a new parameter, click the **parameter value** option button.
   a) Click **[...]** to browse to and select a parameter.

b) If no parameters exist, click **Next** to open the **Parameter Wizard** and create a new parameter.

5. Check the relevant check boxes to make the verification case-sensitive or to apply it as a script-wide rule. A result variable name is displayed in the **Result variable name** text box.
6. *Optional*: Edit the **Result variable name**.
7. Check the **Require boundary strings** check box. This disables the **occurs** frequency settings.

   Left and right boundaries appear in the **Left boundary** and **Right boundary** text boxes. Boundary strings cannot be edited from this dialog box.
8. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
9. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for data verifications within boundaries:

```
transaction TMain
begin
  ...
  WebVerifyDataBound("ahref=\"", "\"", "main.asp?
from=welcome",
    WEB_FLAG_IGNORE_WHITE_SPACE | WEB_FLAG_CASE_SENSITIVE, 1,
    SEVERITY_ERROR, bVerifyDataBoundSuccess1);
  WebPageUrl("http://myHost/shopit/", "ShopIt - Greetings");
  ...
```

Generating an HTML Data-Digest Verification Function

The `WebVerifyDataDigest` function calculates a Web page digest and then subsequently verifies content against the precalculated digest. In the case of HTML or XML source code, a *digest* consists of a character-frequency table, which is calculated for certain characters in a page-response body.

1. In the menu tree, select the Web page that you want to verify.
2. Open **Source** view.
3. Perform one of the following steps:
   - Choose **Script** > **Verify Page Digest** .
   - Click **Add Verifications** and then click **Verify digest** on the **Workflow - Add Verifications** dialog box.

   A name for the digest is displayed in the **Constant name** text box.
4. *Optional*: Edit the **Constant name** text box.
5. In the **Verify** group box, specify how the verification should be applied:
   - **All characters**
   - **Printable characters**
   - **Alphanumeric characters**
6. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.
7. Click **OK**. The function is then added to your test script.

Code Example

The following sample code (in bold) is generated automatically for data digest verifications:

```
const
  DATA_DIGEST_1     :=
```

```
"\h019200000000E7FBEFFFAEF7FD2AFEFFFF2B000000000000000000000000000
00000004
F06040072000800020038000600060002"
"\h0006000A0035004B0024000E000500010001000100010002000100020002F0
059005A0
05800010003000700030002000300010 0";
"\h120001000100060002000300010015001300020001000100030001000300
300A1002
30047003E00BA002100710056009B0001"
"\h000D0062003C00620079003E000300AE006D00BC0023000E002C0005000E0
00200020
0";

transaction TMain
begin
  ...
  WebVerifyDataDigest(DATA_DIGEST_1, 188, 0, 0, 1,
    SEVERITY_ERROR);
  WebPageUrl("http://lab3/ShopItV60/default.asp", "ShopIt -
Greetings");
  ...
```

Customizing User Data

This section explains how user-input data can be parameterized to better emulate real-world user activity during load testing.

User-Input Data

The visual user-input data customization feature allows you to make your test scripts more realistic by replacing recorded user-input data with parameterized data.

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.

User data customization can be used to:

- Perform functional tests by challenging servers with different values each time certain transactions are executed.
- Simulate realistic user behavior by selecting submitted values based on given probability distributions.

TrueLog Explorer Workflow

User data customization is best performed:

- After session information has been parsed from test scripts
- Before server responses have been verified

User-Data Customization Scenarios

There are five scenarios within which user-data customization is typically employed:

- The posting of HTML forms by means of `WebPageSubmit`

- The posting of XML data by means of `WebUrlPostBin` or `WebCustomRequestBin`
- The posting of data for database applications by means of binding input data to SQL commands and stored procedures for database applications including Oracle OCI, DB2 CLI, and ODBC (by means of `OraSet*`, `Ora8Set*`, and `OdbcSet*`).
- The posting of Oracle Forms control value changes (user input) by means of `OraForms*Set` functions.
- The posting of Citrix user-input actions (for example, mouse events and keyboard inputs) by means of `CitrixKey*` and `CitrixMouse*` functions.

Note: User-data customization is not supported for terminal emulation applications, though input parameters can be manually edited in terminal-emulation test scripts.

In the case of HTML, parameterization is performed via form declarations. Parameterization replaces hard-coded form field values, as declared in the `dclform` sections of test scripts, with random variables.

For XML and database applications, parameterization takes place in the parameters of the above-mentioned function calls, where hard-coded parameter values are replaced with data-driven variables or random variables.

Customizing HTML User Data

Explains how to customize HTML user data based on parameters.

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. Choose **File** > **Add TrueLogs** to load a TrueLog into TrueLog Explorer.

   The TrueLog must be based on a Try Script that has had all static session information removed from it.
2. Click **Customize User Data** on the workflow bar. The **Workflow - Customize User Data** dialog box opens.
3. 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 **Post Data** view

     Post Data 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.
4. 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 **Post Data** view identify form fields that can be customized.
5. On the **Post Data** 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.

6. Click the **Create new parameter** option button and then click **Next** to create a new parameter. The **Create New Parameter** page opens.
7. Click the **Parameter from Random Variable** option button and then click **Next**. The **Random Variable** page opens.
8. 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.
9. Enter a name for the variable in the **Name** text box.
10. *Optional*: Create a new random variable file by clicking **New** in the **File** group box.
11. 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.
12. 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.
13. Specify the new random value to use by selecting one of the following choices:

    - **Per usage**
    - **Per transaction**
    - **Per test**
14. 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.

Customizing HTML User Data With an Existing Parameter

1. Choose **File** > **Add TrueLogs** to load a TrueLog into TrueLog Explorer.

   The TrueLog must be based on a Try Script that has had all static session information removed from it.
2. Click **Customize User Data** on the workflow bar. The **Workflow - Customize User Data** dialog box opens.
3. Click the **Customize user input data in HTML forms** link.
4. Click **Find Next** or **Find Previous** on the **Step through TrueLog** dialog box to browse through all `WebPageSubmit` calls in the TrueLog.

   Note: These are the calls that are candidates for user data customization.

   The **Post Data** page shows the page that contains the HTML form that was submitted by the selected `WebPageSubmit` call. When the cursor passes over a form control, a tool tip shows the name of the control in addition to its initial and submitted values.

Note: Highlighted HTML controls on the **Post Data** page identify form fields that can be customized.

5. On the **Post Data** 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.

6. Select **Use existing parameter**.

   This allows specification of an existing parameter that is defined in the `dclparam` or `dclrand` sections of the test script.

7. From the list box, select the datatype to use.

   This list shows all the variables that have been defined in the `dclparam` and `dclrand` sections of the test script.

8. Click **Finish**. The test script will be modified with the new parameter.
9. Click **Find Next** on the **Step through TrueLog** dialog box to locate the next form field to be customized.

Form Data View

For each function call that changes input data, you can customize input data from both the **Post Data** view and the **Form Data** view. The **Form Data** view offers a convenient means of viewing and customizing all customizable form fields. The **Form Data** view is a representation of the forms section of the BDL script with all included name/value pairs.

Right-click a control and choose **Customize Value** (just as with **Post Data** view) to open the **Parameter Wizard**.

Controls for which user data has already been customized are displayed with a green outline.

Multi-Column Data Files

Parameterization from multi-column data files is a powerful means of parameterizing data. It defines files in which specific combinations of string values are stored. Each column in a data file corresponds to a specific parameter. Multi-column data files enable a data-driven test model and allow you to cover all user-data input with a single data file.

Working With Database Applications

This section explains how to apply TrueLog Explorer script-customization features to the testing of applications that rely on common database APIs.

Working With Database Applications - Overview

Silk Performer offers enhanced database support, including enhanced support for Microsoft ODBC and Oracle Call Interface 8 (OCI8), the standard interface for accessing Oracle databases (versions 8 and 9).

Silk Performer database support includes the following APIs:

- Oracle OCI 7.0
- Oracle OCI 8.0
- Microsoft ODBC
- IBM DB2/CLI

For database applications, TrueLog Explorer offers the same script customization features that are available for Web-based applications, including:

- User input-data customization
- Verification functions
- Customization of output-input correlations (comparable to session-handling customization in Web applications)

Note: TrueLog On Error Analysis for databases is not currently supported.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

Sample Database Application - Customer OCI

Customer OCI, the sample database application, simulates a simple application that allows you to add, edit, and delete customer contact information.

Database TrueLog Structure

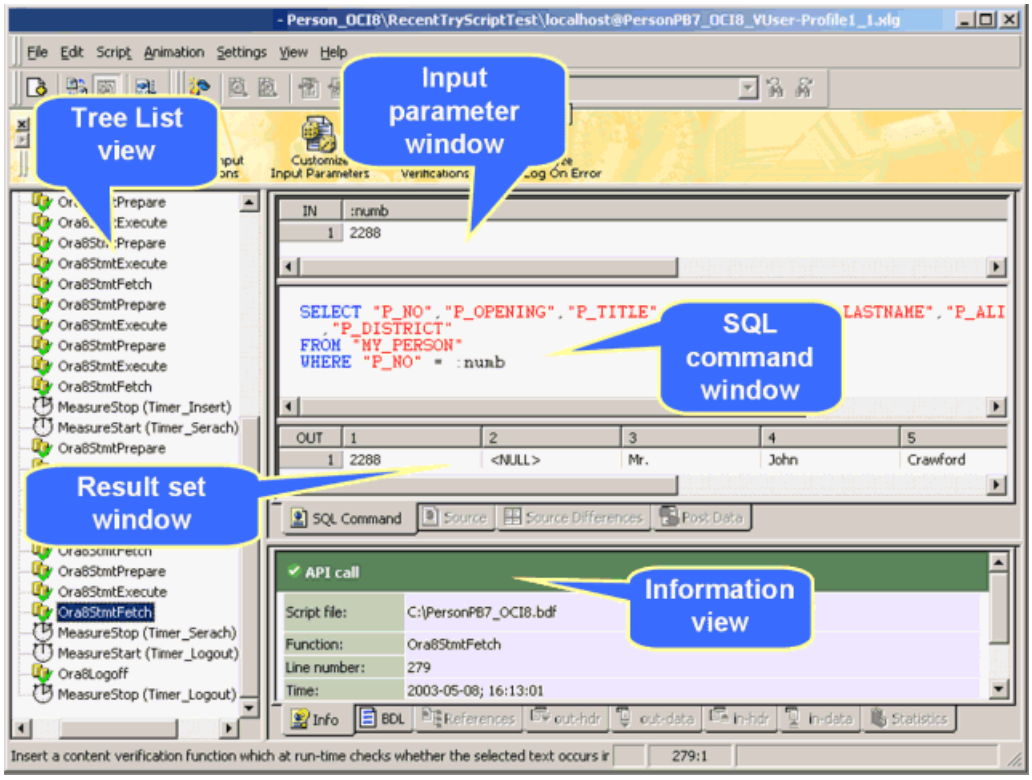
TrueLog Explorer supports the visualization of database requests and responses in the same way that it supports the visualization of HTTP client requests and HTTP/HTML server responses for Web applications. Although with database TrueLogs, a readable database format is presented rather than a GUI as is the case with HTML TrueLogs.

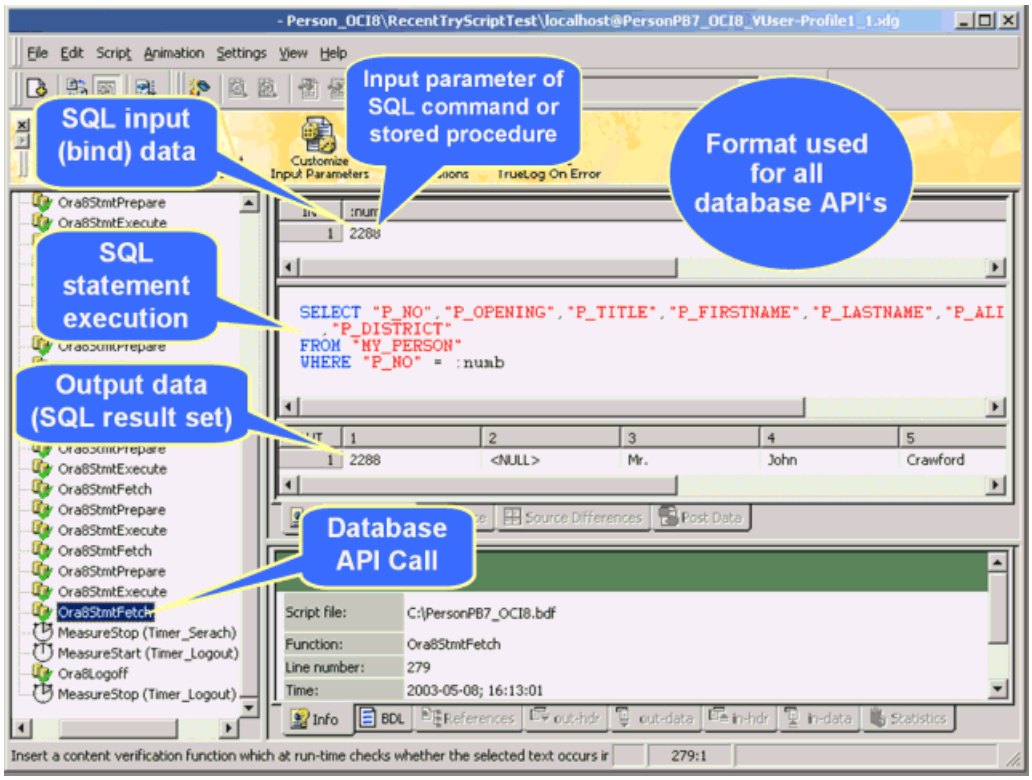
TrueLog Explorer acts as a database browser in which SQL statements and their corresponding input parameters and output parameters can be viewed concurrently. Each supported database API presents data in the same TrueLog format.

The five windows that are offered with database TrueLogs are as follows:

- Menu tree - Lists all relevant database API calls (those that send or retrieve data).
- **Input parameter** - SQL input (bind) data sent to the database.
- **SQL command** - Corresponding SQL command.
- **Result set** - Output SQL data received by the client.
- **Information** - Data regarding the test run.

The only view tabs that are active and applicable to database TrueLogs are **SQL Command**, **Info**, and **BDL**.





Correlations

This section explains how to identify correlations between database input and output parameters.

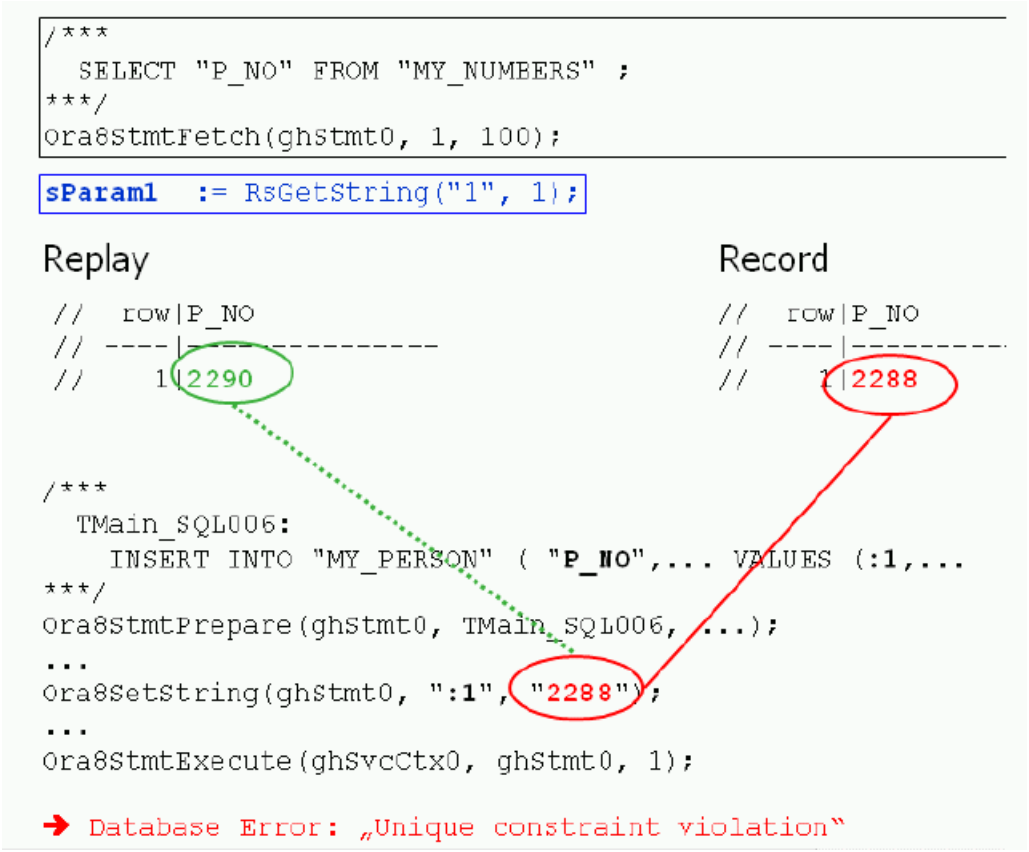
Output-Input Correlations

TrueLog Explorer output-input correlation allows you to step through record and replay API calls side-by-side. It also identifies correlations between database input and output parameters. Similar to session-handling customization for Web applications, output-input correlation compares replay test runs to record sessions to identify differences that may indicate session-relevant values. The differing values are then used as the basis for searches of correlations between output and input values within test-run scripts. Ultimately, TrueLog Explorer enables you to modify identified dynamic values in your test scripts.

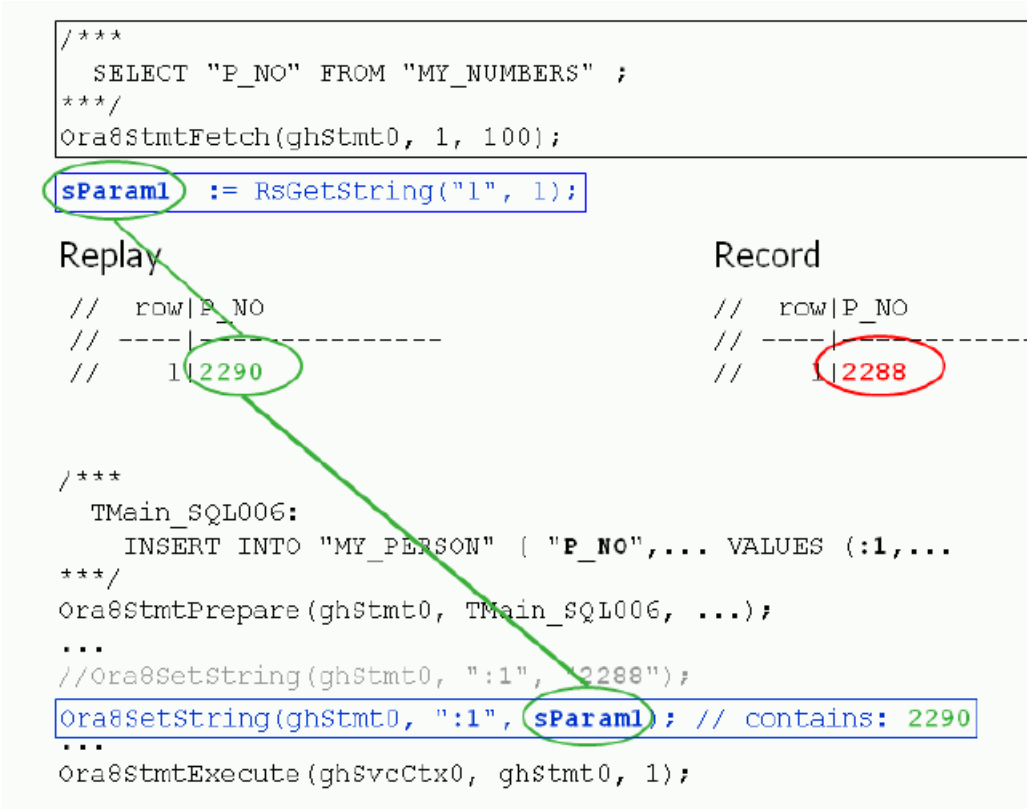
The output-input correlation feature is accessible from the workflow bar when database TrueLogs are open in TrueLog Explorer.

Database-error types are specific to the database type under test. Unique constraint errors typically result when identical data is submitted to a database multiple times. Such errors commonly present themselves when unique values that are used as primary keys in database tables, such as user IDs and order numbers, are submitted. When such personal information is submitted more than once, errors are raised. This type of duplication is the sort of session information that can be identified in test scripts and customized using TrueLog Explorer.

An output-input correlation is illustrated in the following two BDL code examples. The input parameter of the execution call is 2288. Because this value occurred in an output parameter table of a previous call, a correlation has been identified.



The value of the output parameter (2290) is parsed into a variable (`sParam1`) so that it can be used to set the value of the input parameter in future test runs.



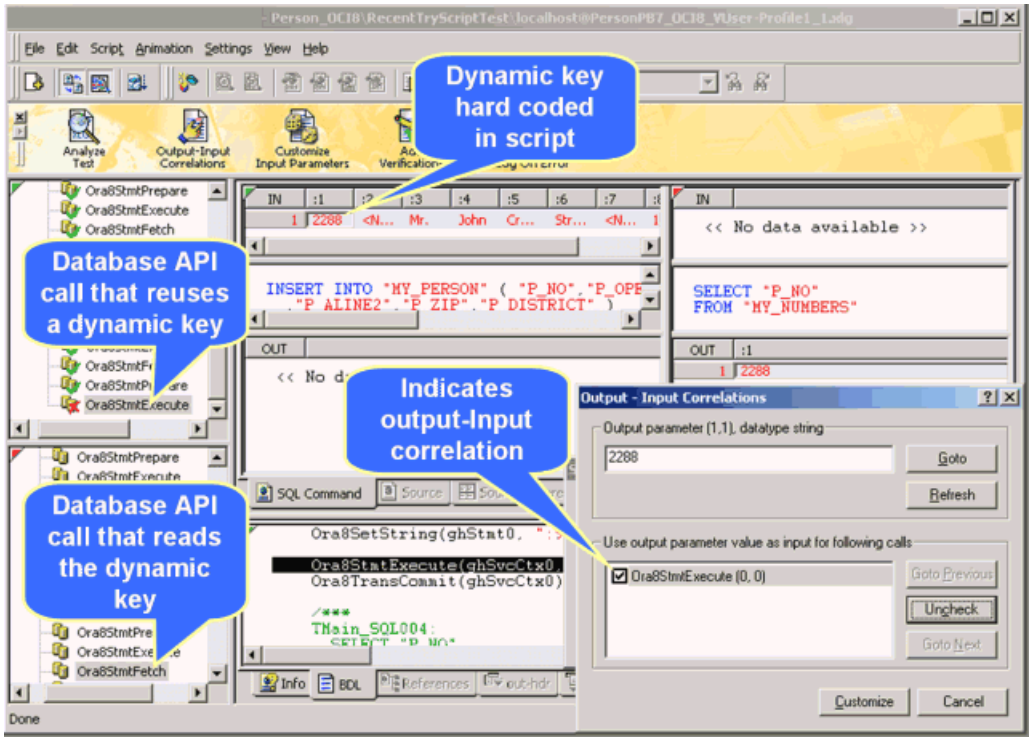
Replacing Session Data with Variables

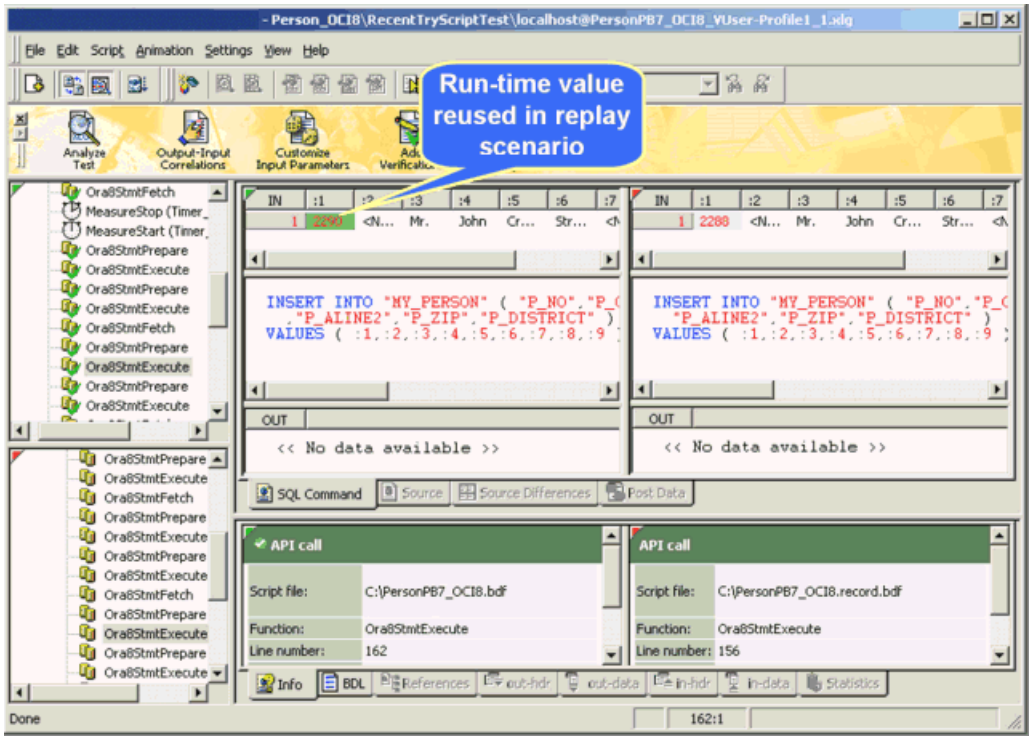
1. Choose **File** > **Add TrueLogs** . The **Open** dialog box opens.
2. Select a replay database TrueLog and click **OK**. The replay TrueLog opens in TrueLog Explorer.
3. Click **Output-Input Correlations** on the workflow bar. The **Workflow - Output-Input Correlations** dialog box displays.
4. Click **Find Differences**. A message box asks if you want to have the associated record TrueLog opened automatically.
5. Click **OK**. Compare mode is enabled. The record TrueLog is opened and the **Step through Correlations** dialog box displays.
6. Using the default selections (**Different result sets** and **Record TrueLog value**) click **Find Next** to advance to the first difference between recorded result set data and replayed result set data.

   Differences are dynamic data returned by the database server, which are potential candidates for correlation. If dynamic data are used in further SQL commands or stored procedures as input parameters, then they constitute correlations between output and input data.
7. Click **Correlate**. The **Output - Input Correlations** dialog box opens.

   If a dynamic value occurs as an input value in another SQL command or stored procedure, the statement/procedure name will appear in the **Use output parameter value as input for following calls** window. TrueLog Explorer can create a variable to subsequently replace the input parameter values of these calls with output parameter values.

   Note: If `No correlating values` appears in the **Use output parameter value as input for following calls** window, then no correlations have been found. Click **Cancel** to return to the **Step through Correlations** dialog box.
8. Click **Check** to select the identified statement name (there may be several).
9. Click **Goto** to see the input value in the context of the test script.
10. If you believe that the input value should be customized, click **Customize** to modify the input parameters in the test script by replacing them with variables.





Initiate a Try Script run to confirm that the script runs without error.

Manual Correlation

Manual correlation is only recommended for applications with which you have advanced knowledge. To perform manual correlation, right-click a replay parameter in the **SQL Command** view and choose **Find Correlation**.

Manual correlation is available in **in-data** view (input parameters), **out-data** view (result sets), and **SQL Command** view. When used in **in-data** view, TrueLog Explorer searches for correlating values (in **out-data** view) that precede the SQL command. When used in **out-data** view, TrueLog Explorer searches for correlating values (in **in-data** view) that follow the SQL command. When used in **SQL Command** view, TrueLog Explorer searches for correlating values (in **out-data** view) that precede the SQL command, in case parameters are hard-coded in SQL commands.

For example:

```
INSERT INTO MY_PERSON (P_NO, P_FIRSTNAME, P_LASTNAME)
VALUES ( 2289, "Michael", "Smith" )
```

Database Parsing Function

For database TrueLogs, TrueLog Explorer offers a single parsing function, `Parse Element Value`. This parsing function is available from the context menu in the SQL window's **Output data** view.

The function generates a parsing statement for one element of a result set. Example:

```
Ora8StmtExecute(ghSvcCtx0, ghStmt0);
Ora8StmtFetch(ghStmt0, ORA_FETCH_ALL, 100); // rows fetched: 1
sParam2 := RsGetString("1", 4);
Print("sParam2: " + sParam2);
```

Depending on the data type (number, float, or string), the appropriate parsing function is then generated (`RsGetInt`, `RsGetFloat`, or `RsGetString`).

Input Parameter Customization

For effective load testing, it is recommended that parameterized input data be assigned to SQL statements so that values are submitted each time database tests are run. Otherwise, identical recorded input parameter values will be sent with each test run, which is not a realistic simulation of real-world user activity. TrueLog Explorer enables you to visually generate code in your test scripts that substitutes recorded input parameter values with customized input parameters.

Note: The same method is used with the customization of user data for Web applications.

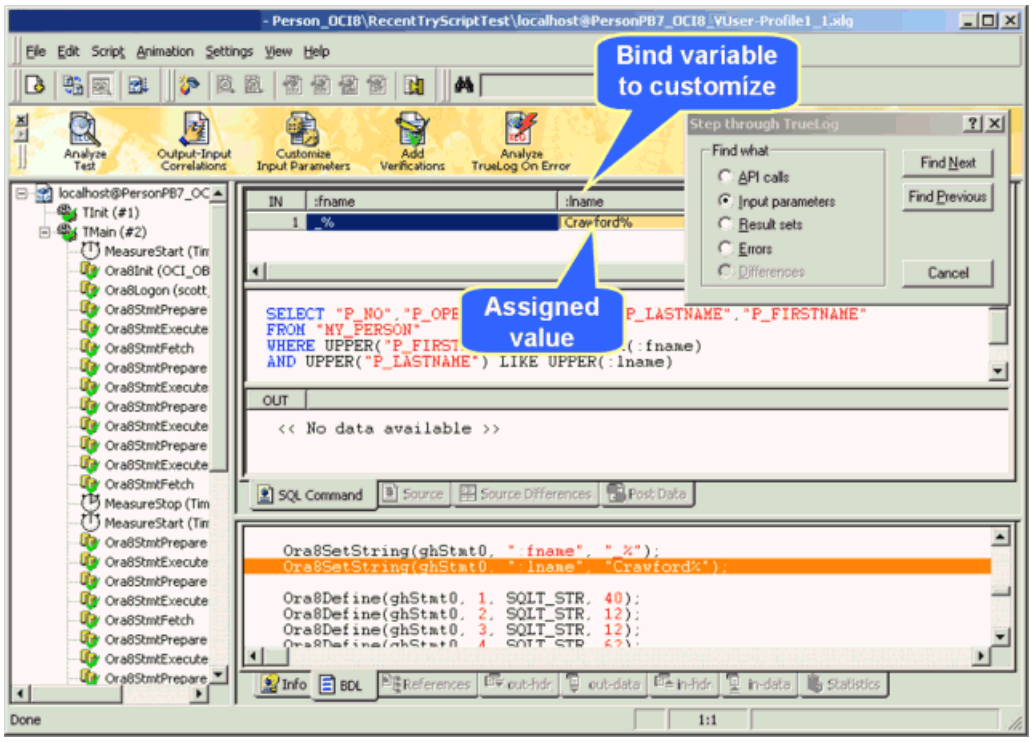
You can use Customer OCI, the sample database application, to experiment with the customization of user-input data. Scripts that replay simple recorded business transactions can be customized with variables that input randomized data into otherwise constant data fields. Input-data values may be constant values, random values, or values drawn from preconfigured data files, such as CSV files.

Creating an Input Parameter Based on a Multi-Column Data File

Parameterization from multi-column data files is a powerful means of parameterizing data. It defines files in which specific combinations of string values are stored. Each column in a data file corresponds to a specific parameter. Multi-column data files enable a data-driven test model and allow you to cover all user-data input with a single data file.

1. Choose **File** > **Add TrueLogs** . The **Open** dialog box opens.
2. Select a replay database TrueLog and click **OK**. The replay TrueLog opens in TrueLog Explorer.
3. Click **Customize Input Parameters** on the workflow bar. The **Workflow - Customize Input Parameters** dialog box opens.
4. Click **Customize input parameters**. The first API node in the test script that includes input parameters is selected and the **Step through TrueLog** dialog box opens with the **Input parameters** option button selected.

   Note: If you do not want to customize input parameters for the selected node, click **Find Next** to advance to the next node that incorporates input parameters.



5. In **Input Parameter** or **SQL Command** view, right-click an input parameter and choose **Customize Value**.

   Two methods are available for modifying values in scripts. You can use an existing parameter that has previously been defined in the `dclparam` or `dclrand` sections of the BDL script, or you can create a new parameter. New parameters can be created from constant values, random variables, or variables based on data in multi-column data files.

   The **Parameter Wizard - Customize Value** dialog box opens.
6. Click the **Parameter from Multi-column Data File** option button and click **Next**. The **Parameter Wizard - Parameter from Multi-column Data File** page opens.
7. Perform one of the following steps:
   - Select a preconfigured multi-column data file from the **File name** list box.
   - Create a new multi-column CSV (Comma Separated Values) file by clicking **New file**. Then, enter a name for the new file and click **OK**.
8. Right-click columns to edit column names, add/delete rows, or add/delete entire columns.

   Note: Each column represents a separate parameter that is assignable to a single input value.
9. Type data into the data fields as appropriate.

   Alternatively, you can select column header names directly to customize them.

10. Customize **Handle name** or the **Parameter name** as required.
11. Click **Next** to save the file. The **Choose the Insert Attribute of the Parameter** page opens.
12. In the **Row selection order** group box specify the order in which the values from the multi-column data file are to be assigned:
    - **Random** - Using function `FileGetRndRow`, the parameter will be assigned from a random row in the data file.
    - **Sequential (machine-wide)** - Using the functions `FileCSVLoadGlobal` and `FileGetNextRow`, the parameter will be assigned from the next sequential row in the data file, on a machine-wide basis. All virtual users on a machine will use the same global row pointer for the current row.
    - **Sequential (test-wide)** - Using the functions `FileCSVLoadGlobal` and `FileGetNextUniqueRow`, the parameter will be assigned from the next sequential row in the data file, on a test-wide basis. All virtual users across all machines will use the same global row pointer for the current row. Check **Remove used data rows** to automatically remove data from the data file, once it has been used.

      
      Note: The removed data cannot be restored.
13. In the **Attribute** group box, specify the frequency at which new insertions should be inserted.
    For example, specify if a new username should be inserted with each transaction or only once per test.
    - **Per transaction**
    - **Per test**

    All available user groups and transactions are available from the **Per transaction** list boxes.
14. Click **Finish**. TrueLog Explorer modifies the test script and saves the multi-column data file.

Customized input parameters are indicated in green.

Initiate a Try Script run with the random variable function in your test script to confirm that the script runs without error.

Verifications for Result-Set Data

TrueLog Explorer allows you to apply verifications to the result sets of database operations such as SQL queries. Verifications enhance your test results by raising errors when anticipated test results are not returned during test runs. By selecting output data and applying simple commands, all required verification functions can be generated and automatically inserted into your BDL scripts.

TrueLog Explorer offers two database operation verification options:

- Verification of element values
- Verification of result-set row counts

Verifying a Database Operation Element Value

1. Choose **File** > **Add TrueLogs** . The **Open** dialog box opens.
2. Select a replay database TrueLog and click **OK**. The replay TrueLog opens in TrueLog Explorer.
3. Choose **Edit** > **Step through TrueLog** .

   Alternative: Click **Step through TrueLog** on the toolbar.

   The **Step through TrueLog** dialog box opens.
4. Click the **Result sets** option button.
5. Click **Find Next** to advance to the first database command that returns a result set.
   For example, you might want `fetch` commands, such as `Ora8StmtFetch`.
   The first data element in the set is selected.
6. Perform one of the following steps:

- Right-click a data element to verify and choose **Verify Element Value**.
- Click **Add Verifications** on the workflow bar and click **Verify the element value**.

The identified data element and its row or column specifications display in the **value** text box, and the **value** option button is selected.

7. *Optional*: To verify against an existing parameter or a new parameter, click **parameter value**.
   a) Click **[...]** to browse to and select a parameter.
      If no parameters exists, the **Parameter Wizard** opens, enabling you to create a new parameter.
   b) Click **OK**.
8. In the **Severity** group box, specify the severity that is raised if the verification returns a negative result.
9. Check the appropriate check boxes to make the verification case-sensitive or apply it as a script-wide rule. A result variable name is displayed in the **Result variable name** text box.
10. Click **OK**. The function is added to the test script.

Initiate a Try Script run to confirm that the script runs without error.

Verifying a Database Result Set Row Count

Result set row-count verifications confirm that returned database result-sets include the specified number of rows. Errors are raised if returned result sets have less or more than the specified number of rows.

1. Choose **File** > **Add TrueLogs** . The **Open** dialog box opens.
2. Select a replay database TrueLog and click **OK**. The replay TrueLog opens in TrueLog Explorer.
3. Choose **Edit** > **Step through TrueLog** .
   Alternative: Click **Step through TrueLog** on the toolbar.
   The **Step through TrueLog** dialog box opens.
4. Click the **Result sets** option button.
5. Click **Find Next** to advance to the first database command that returns a result set.
   For example, you might want `fetch` commands, such as `Ora8StmtFetch`.
   The first data element in the set is selected.
6. Perform one of the following steps:

   - Right-click a data element to verify and choose **Verify Result Set Row Count**.
   - Click **Add Verifications** on the workflow bar and click **Verify the row count**.

   The row count of the current database result is displayed in the **value** text box, and the **value** option button is selected.
7. *Optional*: To verify against an existing parameter or a new parameter, click **parameter value**.
   a) Click **[...]** to browse to and select a parameter.
      If no parameters exists, the **Parameter Wizard** opens, enabling you to create a new parameter.
   b) Click **OK**.
8. Specify whether or not the row count must match `exactly`, `at least`, or `at most` the current row count.
9. In the **Severity** group box, specify the severity that is raised if the verification returns a negative result.
10. Click **OK**. The function is added to the test script.

Initiate a Try Script run to confirm that the script runs without error.

Working With XML Applications

This section describes the script-customization and TrueLog-analysis features that are provided for the testing of XML-based applications.

Working With XML Applications - Overview

TrueLog Explorer offers the same support for XML-based applications that it does for HTML-based applications, including:

- Session handling
- Input-data customization
- Verification functions
- TrueLog On Error analysis

The TrueLog Explorer Web-service support features an intuitive XML data interface that facilitates interaction and analysis of XML TrueLogs.

The TrueLog Explorer XML API works comparably to its Web API. XML data access functions for querying and verifying hold an advantage over Web functions because they allow you to specify the exact locations of values, even when values are specified multiple times. The Web API only allows for the specification of left/right boundaries using `WebParseDataBound` functions that are not flexible enough to adapt to server data changes.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

Sample Web Service

The sample scripts included in this chapter are based on SOAP/XML messages taken from the sample Web service. The scripts were generated using .NET Explorer and exported for use with Silk Performer.

To view the sample Web service using .NET Explorer, navigate to: *http://demo.borland.com/BorlandSampleService/BorlandSampleService.asmx?WSDL*

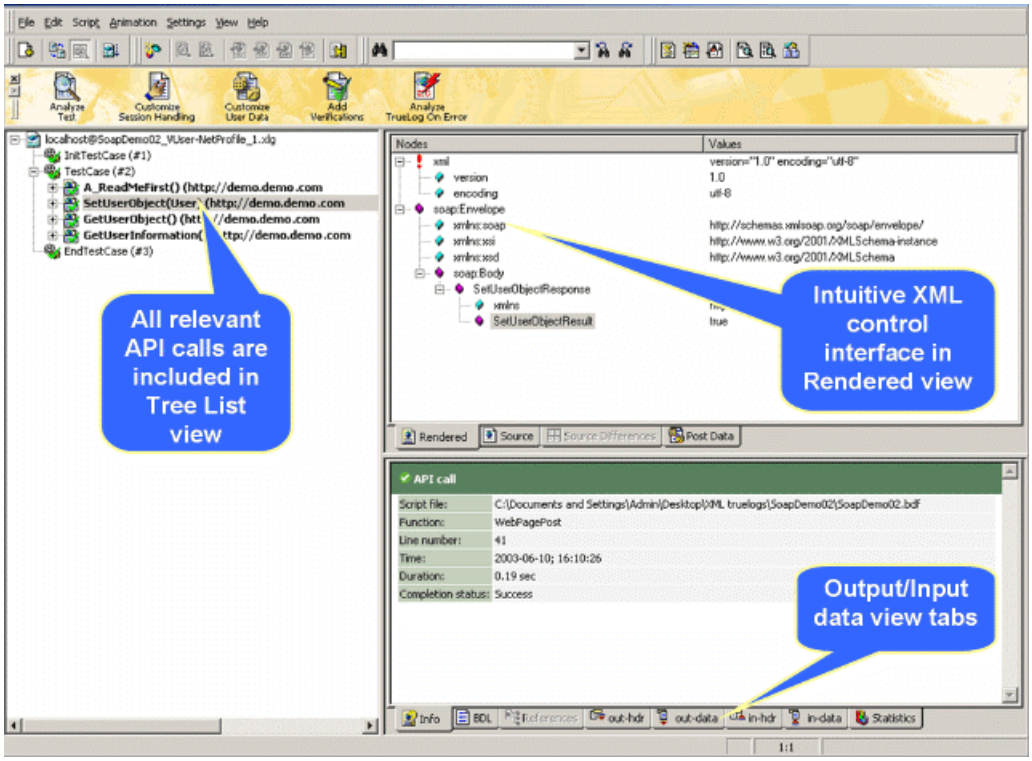
XML TrueLog Structure

XML TrueLogs support the visualization of XML elements, attributes, and values via a readable XML menu tree presentation.

UI Overview of XML TrueLogs

XML TrueLogs support the visualization of XML elements, attributes, and values via a readable XML menu tree presentation. This XML control displays the response data of selected API calls in an intuitive XML interface on the **Rendered** page. The **Source** and **in-data** pages present the raw data that is returned by the server. **Out-data** presents the raw data that is sent to the server.

See the following topics for details on the three main elements of an XML TrueLog.



Menu Tree

The menu tree, located on the left side of the interface, allows you to expand/collapse each TrueLog API node (via double-click). Click a node to display its contents on the Content pane and its history details on the **Information** page.

To view details about a Web page that is included in a TrueLog, select the page's URL or link description in the menu tree. The content is then displayed on the Content pane and its properties are displayed on the **Information** page.

Content Pane

TrueLog Explorer provides multiple views of all received data. Depending on the type of application under test (Web, XML, database, TCP/IP, or UDP), the **Content** pane offers different viewing options:

Item	Description
Rendered (*HTML & XML*)	The **Rendered** page displays the visually rendered response from the server for the selected API node in the menu tree. HTML responses are rendered by an Internet Explorer control running behind the **Rendered** page. XML responses are displayed by an XML control that displays XML data in a menu tree format. Responses that are neither HTML or XML are not rendered and must be viewed using either the **Source** page or the **in-data** page. You can right-click elements and attributes in the XML control to expand the menu tree by one or more levels.
SQL Command	*Suppressed for XML*
Source (*HTML, XML, TCP/IP, and UDP only*)	Presents raw source data. The **Source** page displays the HTML code that is used to generate Web content. It contains the same information that the **in-data** page

Item	Description
	includes. Using the **Source** and **out-data** pages, you can view the in-data and out-data of requests at the same time.
Source Differences (*HTTP based applications*)	The **Source Differences** page lists the in-data (source) differences between selected nodes. This page is helpful in finding dynamic information in server responses. It is the first place to look for session information. Only applicable in **difference mode**.
Post Data (*HTML & XML*)	Displays the request data returned by the server. It displays posted XML data in a menu-tree formatted XML control. Elements are displayed in magenta. Attributes are displayed in cyan.

Information Pane

The **Information** pane offers data regarding testing scripts and test runs, including BDL scripting, HTTP headers, and timing statistics. Depending on the type of application under test (Web, XML, database, Oracle Forms, SAPGUI, Citrix, TCP/IP, UDP, or other), the **Information** pane offers different views.

Item	Description
Info	The **Info** page displays general information about the open TrueLog file and the selected API node, including: • Script file name • Function • Line number • Time • Duration • Absolute URL • Completion status • Additional information, if available
BDL	The **BDL** page displays the BDL script that corresponds to the open TrueLog. The BDL script is automatically positioned to the line of the selected API node.
References	*Suppressed for XML*
out-hdr *(HTTP)*	The **out-hdr** page contains the exact HTTP header that the application (Silk Performer in the case of replay TrueLogs, the browser in the case of record TrueLogs) sends to the server.
in-hdr *(HTTP)*	The **in-hdr** page contains the exact HTTP header that the server sends to the application.
out-data *(HTTP, TCP/IP, UDP)*	Contains data sent with `HTTP-POST` commands from the application to the server.
in-data *(HTTP, TCP/IP, UDP)*	The **in-data** page contains data received by the application from the server. The data is presented in raw format as it is received from the server (no rendering for HTML, no menu tree representation for XML).
Statistics *(HTTP)*	The **Statistics** page shows timing statistics for Web pages. It includes timing information for each Web page component and communication element. It shows exact response times, including the composition for each individual component, in graphical view. This assists you in

Item	Description
	pinpointing the root causes of errors and slow page downloads. The **Statistics** page includes the following data for each page component: • DNS lookup time: The time taken to resolve an IP address from the domain / host name supplied • Connect time: The time taken for the simulated user to connect to the server • Net round trip: The time from the first byte of the client request until the last byte of the server response, including all documents but not embedded objects. • Cache statistics

XML Parsing Functions

XML parsing functions can be applied in a structured manner to XML content returned by servers. XML parsing functions parse out the values of specific elements and attributes within XML documents.

Note: XML parsing is recommended only for enhanced verifications and customizations, not session-handling customization.

Elements and attributes are accessed via XML Path Language (XPath), a standardized language for addressing elements within XML documents. XML parsing functions are applied on the **Rendered** page. Web API style parsing can be executed against XML via the **Source** page.

XML parsing functions include:

- `WebXmlParseNodeValue`
- `WebXmlParseNodeAttribute`

Note: Hierarchical structured values can not be parsed out using the XML control.

Customizing Session Handling for XML Applications

Normally session information is not included in XML code, so XML session-handling customization is rarely required.

In those rare instances where session information is included in XML code, proceed with the session-handling customization process outlined for Web-based applications.

User-Input Data Customization

The goal of user-input data customization is to customize the values of submitted XML data elements and attributes. Data that may, for example, come from recorded SOAP traffic.

With HTML, hard-coded form-field values, as declared in the `DCLFORM` sections of test scripts, are replaced with random variables. In such instances, parameterization takes place in `form` declarations.

With XML, hard-coded element and attribute values, as declared by the post data parameters of corresponding functions, are replaced with random variables. Parameterization takes place in the `post data` parameters of `WebUrlPostBin`, `WebCustomRequestBin`, and `WebPagePost` calls (`WebPagePost` is generated by .NET Explorer. It is the page-level API call used for `HTTP-POST` commands).

User-input data customization can be used for the following:

- Perform functional tests by challenging servers with different values each time certain transactions are executed.

- Simulate real-world user behavior by selecting submitted values based on given probability distributions.

Note: To set up parameterization for the values of submitted XML data elements and attributes, proceed with the user-input data process outlined for Web-based applications.

Verification Functions for XML Applications

XML verification functions verify the values of specified XML elements and attributes (which are identified by XPath queries) during test runs.

Verification Functions for XML Applications

XML verification functions verify the values of specified XML elements and attributes (which are identified by XPath queries) during test runs. XML verification functions are inserted by right-clicking within the **Rendered** page.

TrueLog Explorer offers the following XML verification functions:

- `WebXmlVerifyNodeValue` - Checks a selected element's value. This function is generated when `Verify Element Value` is selected from the XML tree-control context menu.
- `WebXmlVerifyNodeAttribute` - Checks a selected attribute's value. This function is generated when `Verify Attribute Value` is selected from the XML tree-control context menu.

Verification Checks During Replay

Verification functions can be enabled/disabled via Silk Performer profile settings. From within Silk Performer, choose **Settings** > **Active Profile** , then choose **Replay** > **Web** > **Verification** > **HTML** > **XML** .

The **XML Verification** check box enables/disables the functions `WebXmlVerifyNodeValue` and `WebXmlVerifyNodeAttribute`.

Profile settings in scripts can be overridden using the `WebSetOption` BDL function.

Inserting an XML Verification Function

All parsing and verification functions must be specified before the Web API calls that initiate the parsing/verification of response data. Multiple parse/verification functions can be specified before each Web API call.

Note: The order in which parsing/verification functions are inserted into a test script is not relevant.

1. Select and right-click an XML element or attribute on the **Rendered** page and choose **Verify Element Value**.

   Alternative: When verifying an attribute, the context menu instead offers the **Verify Attribute Value** command.

   The **Insert XML Verification Function** dialog box opens. The selected element value is pre-loaded into the **value** edit box and the **value** option button is selected.
2. To verify against a parameter, click the **parameter** option button.

   In addition to being able to verify against a constant value, you can also verify against either an existing parameter or a new parameter.

   a) Click **[...]**.

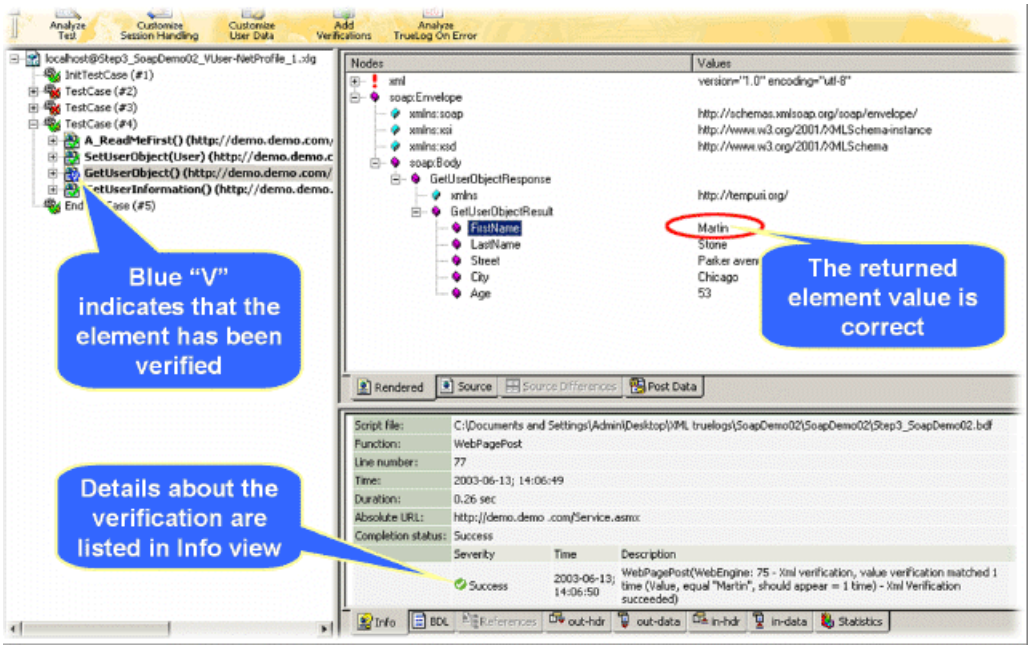
   - If a parameter already exists, you can browse to and select the parameter.
   - If no parameters exist, click **[...]**. The **Parameter Wizard** opens enabling you to create a new parameter.
3. Specify the frequency by which the selected element is to appear:

- Select `exactly`, `at least`, or `at most` from the **occurs** list box.
- Enter a number in the **time(s)** field.

Note: The settings on this dialog are automatically set to values that guarantee a successful verification for the current page. Only in cases in which you want to make the verification more tolerant should these settings be changed (for example, by changing `exactly 2` times to `at least 1` time, or by making a verification case-insensitive).

4. Check the **exact XPath query** check box if the element is to be verified only in the specified XPath.
5. Specify whether or not the verification is to be applied as a script-wide rule and/or case sensitive by checking the appropriate check boxes. A pre-loaded result variable name appears in the **Result variable name** text box; edit this name as required.
6. In the **Severity** portion of the dialog, specify the severity that is to be raised if the verification returns a negative result (`Error`, `Warning`, `Informational`, or `Custom`).
7. Click **OK** to add the function to your test script.

   XML attribute verification is identical to XML element verification, except that an attribute name must also be specified.
8. Initiate a Try Script run to confirm that your script customization runs accurately.



Working With SAPGUI Applications

For SAPGUI-based applications, TrueLog Explorer offers TrueLog recording, TrueLog replay, basic TrueLog analysis, data parsing, content verification, and user-input data customization.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

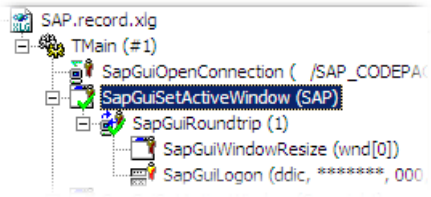
SAPGUI TrueLog Structure

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.

SAPGUI TrueLog Functions

TrueLog Explorer offers two high-level SAPGUI functions:

- `SapGuiActiveSetWindow` – These are top-level API nodes that indicate the generation of a new GUI window. All actions taken on windows are grouped below a `SapGuiActiveSetWindow` function.
- `SapGuiRoundTrip` – These are virtual nodes; there are in fact no API calls called `SapGuiRoundTrip` that are sent to the server. These nodes are used to group all client-side actions that occur in the course of a server round-trip. Both the before and after states of round-trips can be viewed. Multiple round-trip nodes may be included under each `SapGuiSetActiveWindow` node.



Stepping Through SAPGUI TrueLogs

Following tests, it is typical to have multiple TrueLog On Error files open in TrueLog Explorer (one TrueLog for each virtual user who returns an error). You can jump from one error to the next sequentially as they occurred in time. This feature simplifies the process of analyzing errors because there is no need for you to manually review all open TrueLogs to find the next error in a sequence.

1. After the completion of a load test, click Silk Performer **Explore Results**.

   Note: TrueLog On Error files are generated only when the Silk Performer **Generate TrueLog On Error** option is enabled.

   The **Explore Results** dialog box displays.
2. Click **TrueLog Explorer**.

   Note: The **TrueLog Explorer** button is disabled when either no errors are detected during a test or when Truelog On Error is not enabled. In such instances, proceed directly to analyzing results with Performance Explorer.

   TrueLog Explorer launches with the TrueLog On Error files that were generated for the current test, and the **Step through TrueLog** dialog box displays.
3. In the **Step through TrueLog** dialog box, select the appropriate option button. You can advance through:

   - `All API calls` (each API node)
   - `All Roundtrips` (`SapGuiRoundtrip` calls)
   - `All Windows` (`SapGuiSetActiveWindow` nodes)
   - `All Customizable calls` (the next node in which field value changes can be customized)
   - `All errors`

   TrueLog On Error files are searched sequentially, as they are recorded in time.
4. Click **Find Next** to advance to the first API call, roundtrip, window, customizable call, or error.

Error messages are displayed in the Info pane. Each API node that contains a replay error is tagged with a red “X” in the **TrueLog** menu tree.

Analyzing SAPGUI Test Scripts

Following the generation of a test script, you can determine if the script runs without error by running a Try Script run. A Try Script run determines if a script accurately recreates the intended transactions.

The default option settings for Try Script runs include live display of data downloaded during testing and the writing of log and report files.

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 delay between transactions.

1. Click **Try Script** on the Silk Performer workflow bar. The **Try Script** dialog box displays.
2. To view rendered page transitions during a Try Script run, check the **Animated Run with TrueLog Explorer** check box.
3. Click **Run**.

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

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

   TrueLog Explorer opens, showing you the data that is downloaded during the Try Script run. Each main SAPGUI window accessed during recording is listed as a high-level `SapGuiSetActiveWindow` API node in the TrueLog menu tree. All recorded server round-trips and user actions are listed as subnodes of corresponding `SapGuiSetActiveWindow` nodes.

Replay and Record TrueLogs

This section explains how to analyze differences between SAP record and replay TrueLogs.

Comparing Replay and Record TrueLogs

When testing SAPGUI applications, differences can occur in corresponding control values between replay and record TrueLogs. Some differences may result in replay errors.

1. Open a replay Truelog and click **Analyze Test**. The **Analyze Test** dialog box displays.
2. Click **Compare your test**. The associated record TrueLog opens in compare view. The **Step through TrueLog** dialog box displays.
3. In the **Step through TrueLog** dialog box, select the appropriate option button. You can advance through:

   - `All API calls` (each API node)
   - `All Roundtrips` (`SapGuiRoundtrip` calls)
   - `All Windows` (`SapGuiSetActiveWindow` nodes)
   - `All Customizable calls` (the next node in which field value changes can be customized)
   - `All errors`

   TrueLog On Error files are searched sequentially, as they are recorded in time.
4. Click **Find Next** to advance to the first API call, roundtrip, window, customizable call, or error.
5. Visually compare the values and states of the controls and screenshots to see if there are any differences. Based on any differences you discover, complete any required customizations.
6. Click **Try Script Run** to confirm that your customizations run without error.
7. Repeat this procedure as many times as is required until your script is fully customized and all necessary customizations have been added.

8. Turn off compare mode by performing one of the following steps:
   - Choose **View** > **Compare Mode** .
   - Click the **Compare Mode** icon.

Synchronizing Replay and Record 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.

SAPGUI Test-Script Customization

After you generate a test script with Silk Performer and execute a Try Script run, TrueLog Explorer can help you customize the script with the following:

- Content-verification functions – Using the **Add Verifications** tool, you can gain insight into data that is downloaded during tests, enabling you to verify that the content that is to be sent by the server is in fact received. Verifications remain useful after system deployment for ongoing performance management.
- Parsing functions – TrueLog Explorer allows you to insert SAPGUI parsing functions visually in **Source** view and on the **Controls** page. Manual code writing is not required; TrueLog Explorer automatically generates parsing functions in scripts.
- Parameterized input data – With user data customization, you can make your test script more realistic by replacing static user-input data with dynamic, parameterized data that changes with each transaction. Manual scripting is not required to create these data-driven tests.

For each SAPGUI function call that changes input data, you can verify return values, parse values, and customize input data. These operations can be executed from both **Source** view (by right-clicking within a control) and the **Controls** menu tree.

Customizing User-Input Data for a Form Field

Under real world conditions, SAPGUI-application users submit unpredictable combinations of data into forms. One goal of effective SAPGUI application testing is to emulate such irregular and diverse user behavior in test scripts.

You can customize the user-input data that is entered into forms during testing with the **Parameter Wizard**. The Parameter Wizard lets you specify values to be entered into form fields, enabling your test scripts to be more realistic by replacing recorded user input data with randomized, parameterized data.

1. Select a TrueLog in the **TrueLog** menu tree. TrueLog content appears on the **Rendered** and **Information** pages.
2. Choose **Edit** > **Step through TrueLog** .

   Alternative: Click **Step through TrueLog** on the toolbar.

   The **Step through TrueLog** dialog box opens.
3. Click the **Customizable calls** option button and click **Find Next** to step through all form fields in the TrueLog that offer input customization.

Note: Controls that can be customized are outlined in orange. Controls that have already been customized are outlined in green. Controls that are outlined in blue can have their values parsed or verified, but they cannot be customized.

4. On the **Post Data** 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. 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.

Multi-Column Data Files

Parameterization from multi-column data files is a powerful means of parameterizing data. It defines files in which specific combinations of string values are stored. Each column in a data file corresponds to a specific parameter. Multi-column data files enable a data-driven test model and allow you to cover all user-data input with a single data file.

SAPGUI Controls Menu Tree

For each SAPGUI function call that changes input data, you can verify return values, parse values, and customize input data from both **Source** view and the **Controls** menu tree in the information pane. The **Controls** menu tree offers a convenient means of viewing and customizing form fields. The **Controls** menu tree is a visual representation of the form section of the BDL script, with all included name/value pairs.

Controls that can be customized are outlined in orange or blue. Controls that are outlined in orange generally offer value parsing, value verification, and user-input data customization. Controls that are outlined in blue offer value parsing and value verification, but not user-input data customization. Right-click a control and choose the script customization type that you want to execute.

Controls for which user data has already been customized appear with a green outline.

Copy Control ID Function

The `Copy Control ID` function copies the control ID of a selected SAPGUI control to the clipboard. This eases non-visual scripting of special calls within Silk Performer. To copy a control ID, right-click a GUI control and choose **Copy Control ID**. The control ID is then copied to the clipboard.

Note: Controls can be selected either in **Source** view or in the **Controls** menu tree.

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.

Adding a Value Verification

By right-clicking a SAPGUI control that you want to have verified, the required verification function can be generated and inserted into your BDL script. TrueLog Explorer offers one pre-enabled value verification function for SAPGUI applications.

1. In either **Source** view or the **Controls** menu tree, select a control field for which you want to verify the return value.

   Controls that can be verified have blue outlines around them. Most controls that are outlined in orange can also be verified.
2. Right-click within the control field and choose **Verify Value**. The **Insert Value Verification Function** dialog box displays. Use this dialog to specify the type of verification function that should be inserted into the BDL script.
3. From the **Verify that the value of the selected control** list box, select **is equal to** or **is not equal to** (based on your requirements).
4. Specify whether the verification should use a **constant value** or a **parameter value**.
5. Specify whether the verification is to be **Case sensitive** and if white spaces should be ignored.
6. In the **Severity** group box, specify the severity that should be raised if the verification returns a negative result.

7. Click **OK**. The function is then added to your test script.

Once the BDL script has been successfully modified, repeat this process for each verification that you want to add to your BDL script.

Adding a SAPGUI Parsing Function

For any SAPGUI function call in which input data is inserted, you can insert a parse-value function. Value parsing can be useful for enabling replay when session-specific strings must be parameterized to enable accurate replay. For instance, you might parse customer IDs into a parameter that is inserted into a `customer ID` field.

Though toolbars and title bars cannot be parsed, most common controls can be parsed, including text fields, field labels, combo boxes, buttons, and more.

1. In **Source** view or the **Controls** menu tree, select a control field for which you want to parse a value.

   Controls that can be verified have blue outlines around them. Most controls that are outlined in orange can also have their values parsed.
2. Right-click within the control field and choose **Parse Value**. The **Insert Value Parsing Function** dialog box opens and offers settings by which the parsing function can be adjusted.
3. *Optional*: In the **Parameter name** text box, enter a name for the parameter that is to receive the result of the parsing function.
4. *Optional*: In the **Informational statement insertion** area, click **Print statement** to insert an informational print statement into the script. The result of the parsing function will be written to the Silk Performer **Virtual User Output** window.
5. *Optional*: In the **Informational statement insertion** area, click **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 the time measurements of load tests, these files should only be used for debugging purposes and should not be generated for full load tests.
6. Click **OK**. The parsing statement is inserted into your test script.

Once you have customized how your script handles session information and user-input data, you have added all necessary verification functions, and have completed any required manual BDL script editing via Silk Performer, your testing script should run without error.

Working With Oracle Forms Applications

This section explains how to customize an Oracle Forms test script based on the results of a Try Script run.

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.

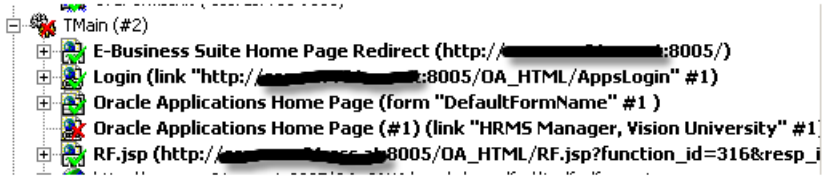
Customizing Oracle Applications 12i Session Information

Ensure that the `icx_ticket` parameter has been customized correctly.

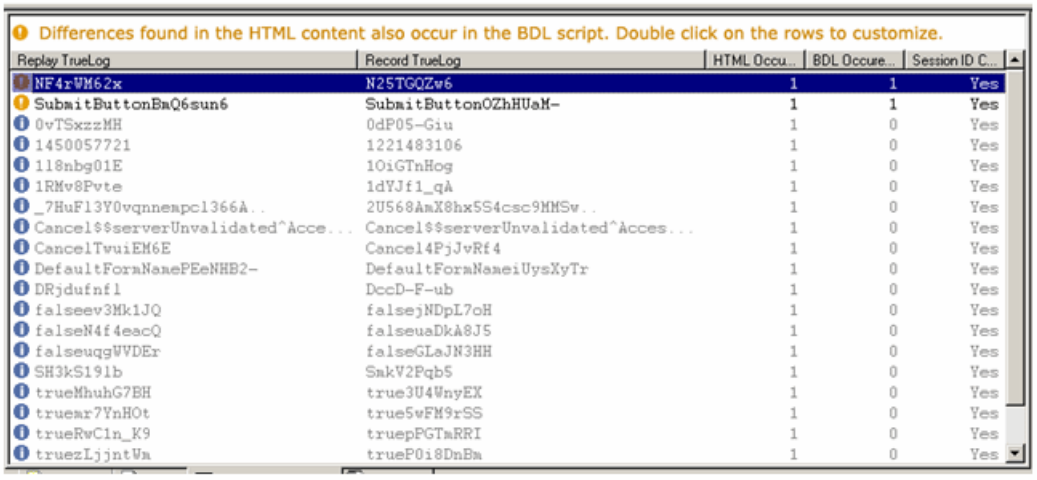
1. Perform a trial run of the script by clicking the **Try Script** button on the **Workflow Bar**.

   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:



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 **Source Differences** tab.



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.

Oracle Forms TrueLog Structure

This section explains the Oracle Forms TrueLog structure and the **Form Controls** window.

Oracle Forms TrueLog Structure - Overview

Oracle Forms applets are commonly incorporated into applications that are built on both the Oracle Forms protocol and the HTTP protocol. When such applications are recorded, the resulting BDL scripts include both Oracle Forms and Web function calls. Oracle Forms TrueLogs support such mixed protocols. The **TrueLog** menu tree lists both the Oracle Forms and HTTP function calls that are included in the active BDL script. Each API node in the menu tree signifies a unique function call.

Oracle Forms TrueLogs feature a **Form Controls** window that displays the elements of the selected Oracle Forms applet window in tabular-data representation (otherwise this binary protocol would be unreadable). For the HTTP calls that call the HTML pages in which Oracle Forms applets are embedded, the **Form Controls** window displays fully rendered HTML.

Oracle Forms TrueLogs have a unique structure. Each `OraFormsSetWindow` call results in a new top-level node. An `OraFormsSetWindow` call is scripted for each window that is activated during a recording session. All actions that are performed on a window (for example, editing of a text control, selecting a list entry, etc.) are shown as sub-nodes of each `OraFormsSetWindow`.

`OraFormsInit` is included in the Init transaction. `OraFormsDestroy` is included in the End transaction. The first Oracle API call in the main transaction is `OraFormsConnect`.

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).

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).

In-Data / Out-Data Pages

The data that is communicated between Silk Performer and an Oracle Forms server can be tracked on the in-data and out-data pages. During Oracle Forms replay, the in-data page displays the data that is received by Silk Performer from the Oracle Forms server. The out-data 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). 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-data/out-data 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:
```

Terminal Messages

Terminal messages are used to end communication round-trips between client and server. There are three terminal-message types:

Terminal 1 messages: This is the most straightforward message-termination method. Terminal 1 messages are always initiated by the client. An example is the client asking the server, “The user just pressed this button, what should I show in the UI?” The server then responds with the requested answer and terminates the exchange with a Terminal 1 message.

Terminal 2 messages: Sometimes, communication between client and server is more complicated. For example, if the client says, “The user just pressed this button, what should I show in the UI?” but the server needs more information from the client before it can provide an answer. The server responds with a Terminal 2 message, asking something like, “What is the state of option button XYZ?” If the client has a prepared, scripted response for this request, it will provide the answer to the server’s question and end communication with a Terminal 2 message. The server will then provide a scripted response to the client’s original question and end the communication with a Terminal 1 message.

This type of exchange can be illustrated as follows:

1. Request from client (Terminal 1, out-data tab)
2. Request from server (Terminal 2, in-data tab)
3. Response from client (Terminal 2, out-data tab)

4. Response from server (Terminal 1, in-data tab)

Terminal 3 messages: Indicates that the server has spontaneously initiated a Get call requesting a value from the client. These are requests that are not initiated by the client. Example: The server says, “I have emptied my cache, please tell me the value of option button XYZ again.” The client then needs to respond with the appropriate answer and terminate communication with a Terminal 3 message.

Terminal 3 messages sometimes result in replay-script breakage. Because these server requests are unanticipated, the client can only provide a default answer, such as null. If the server accepts the default answer provided by the client, for example if the server does not really need a specific answer to its Get call, then replay will continue (this is often the case). If however the server finds the default response to be unacceptable, you must manually script an `OraFormsSet` and an `OraFormsOnMessageGet` function to prepare the correct response for the server’s Get call.

Example:

```
OraFormsSetRectangle("VISIBLERECT", 0, 0, 119, 24,
ORA_SET_TYPE_MESSAGEGET);
OraFormsOnMessageGet("LINE_CUSTOMER_ITEM_DSP_0"); // Requested
Item
OraFormsMouseClick("LINE_CUSTOMER_ITEM_DSP_0", 70, 18, 0); //
Requested Item
```

Terminal -1 messages: Indicates an error in communication between client and server. Replay has failed because the server has received an unacceptable answer to its request.

Analyzing Errors in Oracle Forms Tests

Complete an Oracle Forms load test.

After tests, it is typical to have multiple TrueLog On Error files open in TrueLog Explorer (one TrueLog for each virtual user who returns an error). With the TrueLog Explorer **Find Errors** feature, you can jump from one error to the next chronologically, regardless of which TrueLogs the errors were recorded in. This simplifies the process of analyzing errors because there is no need for you to manually review all open TrueLogs to find the next error in a sequence.

1. Click the Silk Performer **Results** button.

   Note: TrueLog On Error files are generated only when the Silk Performer **Generate TrueLog On Error** option is enabled.

   The **Explore Results** dialog box opens.
2. Click **TrueLog Explorer**.

   Note: The **TrueLog Explorer** button is disabled in the following situations:

   - No errors are detected during a test
   - Truelog On Error is not enabled

   In such instances, proceed directly to analyzing results with TrueLog Explorer.

   TrueLog Explorer launches, with the TrueLog On Error files that were generated for the current test and the **Step through TrueLog** dialog box open.
3. Advance through all **Forms** windows (`OraFormsSetWindow` calls), all **API calls** (each API node), or **all errors**.

   a) Select the appropriate **radio** button on the dialog box.

   TrueLog On Error files are searched chronologically.
4. Click **Find Next** to advance through the TrueLog.

   Error messages are displayed on the **Info** page.

API nodes that contain replay errors are tagged with red “X” marks in the **TrueLog** menu tree.

Analyzing Oracle Forms Test Scripts

Once you have generated a test script you can determine if the script runs without error by performing a Try Script run. A Try Script run determines if a script accurately recreates the recorded interactions.

Default option settings for Try Script runs include live display of data downloaded during testing and the writing of log and report files.

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 delay between transactions.

1. In Silk Performer, click **Try Script** on the workflow bar. The Try Script dialog box opens.
2. Check the **Animated Run with TrueLog Explorer** check box to view page transitions during the Try Script run.
3. Click **Run**.

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

   The Try Script run begins. The **Monitor** pane opens, giving you detailed information about the progress of the run.
4. Silk Performer opens, showing you the data that is downloaded during the Try Script run.

   Each main **Oracle Forms** window accessed during recording is listed as a high-level API node in the TrueLog Explorer menu tree. All recorded actions are listed as subnodes.
5. If any errors occur during the Try Script run, TrueLog Explorer will assist you in locating them and customizing your script.

Comparing Oracle Forms Replay and Record TrueLogs

Open a TrueLog with TrueLog Explorer.

When testing Oracle Forms applications, differences can occur in corresponding control values between replay and record TrueLogs. More severe errors can also result in differences between record and replay TrueLogs (for example, a message box might indicate that a duplicate record has been entered (likely because a required input-data customization has not been performed on the test script).

1. Click **Analyze Test**. The **Workflow - Analyze Test** dialog box opens.
2. Click **Compare your test**. The associated record and replay TrueLogs open in compare view. The **Step through TrueLog** dialog box also opens.
3. Select a criteria by which to search:
   - `Forms windows`
   - `API calls`
   - `Errors`
4. Click **Find Next**. The first set of corresponding Forms windows, API calls, or errors in the record and replay TrueLogs are displayed.
5. Compare the states of the controls to see if there are differences. TrueLog Explorer highlights any name or control value that changes between record and replay. This helps you identify where customizations may be necessary and where verifications may be beneficial.
6. Complete any required customizations (based on differences you have discovered).
   a) Click **Try Script Run**.

   Tests to see if the customizations run without error.
7. Repeat this procedure as many times as is required until all necessary customizations have been added.

8. Click **Compare Mode** to disable compare mode.

Unanticipated Get Calls

The vast majority of server requests are anticipated and accurately handled by recorded scripts. Occasionally during script replay a server spontaneously sends out additional `Get` calls that it did not send out during script recording. In such instances the client can only respond with default answers to the server. If the server accepts a default answer, replay continues without error. If the server finds the default answer to be unacceptable though, you must manually script an `OraFormsSet` and an `OraFormsOnMessageGet` function to prepare the correct response for the server's `Get` call.

For example, the server spontaneously sends a `Get` request asking for the VALUE property of a certain UI control. Because there is no scripted function in the script to handle this `Get` request, the client responds with the default value of `null` for strings. The server does not accept this answer. Connection to the client is aborted and the replay script runs into errors.

To prepare for manual scripting of an additional `Get` function to manage this situation, it is necessary that you analyze the in-data and out-data that is exchanged between the client and server. This analysis can only be conducted if you have set the Silk Performer logging level to `debug`.

Note: If logging level was not set to `debug` before your test was run, after you change the setting you will need to rerun your test to generate the required debug data.

Note: Also, the `record=names` parameter on the Oracle Forms server must be enabled to allow accurate recording of scripts.

The `OraFormsOnMessageGet` function that you script must provide an answer that is different than the default answer for the subsequent `Get` call on the relevant UI element.

Note: It is actually an `OraFormsSet` function that provides the answer in the script (in the line preceding the `OraFormsOnMessageGet` function call).

As the following BDL script example illustrates, when a `Get` call is handled, you have to read the script from bottom to top:

Example

```
OraFormsSetRectangle("VISIBLERECT", 0, 0, 119, 24,
ORA_SET_TYPE_MESSAGEGET);
OraFormsOnMessageGet("LINE_CUSTOMER_ITEM_DSP_0"); // Requested
Item
OraFormsMouseClick("LINE_CUSTOMER_ITEM_DSP_0", 70, 18, 0); //
Requested Item
```

This code sample can be read as: The client will perform a mouse click on the UI element `LINE_CUSTOMER_ITEM_DSP_0`. This is represented by the `OraFormsMouseClick` function. Upon the mouse click, you expect to receive a `Get` call from the server. To prepare for the `Get` call in advance of the mouse click, you insert the `OraFormsOnMessageGet` function. You then expect the server to ask for the property `VISIBLERECT` so you place the property value of type `rectangle` on the stack before the `OraFormsOnMessageGet` function using the `OraFormsSetRectangle` function.

Setting Silk Performer Log Level to Debug

To prepare for manual scripting when an unanticipated `get` call is received, it is necessary that you analyze the in-data and out-data that is exchanged between the client and server. This analysis can only be conducted if you have set the Silk Performer log level to `debug`.

1. In Silk Performer, choose **Settings** > **System** .

2. Click the **Oracle Forms** group button.
3. Select `debug` from the **Log level** list box.

Oracle Forms User-Input Data Customization

Once you have generated a test script with Silk Performer and executed a Try Script run, TrueLog Explorer can help you customize the script with parameterized 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 changes with each transaction. Manual scripting is not required to create such data-driven tests.

Multi-Column Data Files

Parameterization from multi-column data files is a powerful means of parameterizing data. It defines files in which specific combinations of string values are stored. Each column in a data file corresponds to a specific parameter. Multi-column data files enable a data-driven test model and allow you to cover all user-data input with a single data file.

Customizing Oracle Forms User-Input Data

Under real world conditions, Web application users submit unpredictable combinations of data into forms. One goal of effective Web application testing is to emulate such irregular and diverse user behavior using test scripts.

You can customize the user-input data that is entered into forms during testing with the TrueLog Explorer Parameter Wizard. The Parameter Wizard lets you specify values to be entered into form fields, enabling your test scripts to be more realistic by replacing recorded user input data with randomized, parameterized user data.

1. Select a node in the TrueLog menu tree that includes user-data input.

   You can customize the input values of controls that have yellow backgrounds.
2. Right-click in the **Value** column of a control and choose **Customize Value**.

   With the Parameter Wizard, you can modify script values in either of the following 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.
3. Click the **Create new parameter** option button.

   a) Click **Next**

   The **Create New Parameter** dialog box opens.
4. Click the **Parameter from Random Variable** option button.

   a) Click **Next**

   The **Random Variable Wizard** opens.
5. Select the type of random variable you want to insert into your test script from the list box

   A brief description of the highlighted variable type appears in the lower pane.

   a) Click **Next**

   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.
6. Type a name for the variable in the **Name** field.

   a) Select the order in which the values are to be called.

   - Random

- Sequential

7. Select a preconfigured datasource from the **File/Name** list box.

   Alternative: New random variable files can be created by clicking **New**.

8. Select **Per test** random value generation.

   a) Click **Finish**.

   This modifies 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.

Initiate a Try Script run with the random variable function in your test script to confirm that the script runs without error.

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.

Content Verification Functions for Oracle Forms

TrueLog Explorer offers pre-enabled verification functions for Oracle Forms applications. Simply right-click the objects that you want to have verified and choose a verification function. The specified verification function will be generated and inserted into your BDL script automatically.

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.

By comparing replay test runs with record test runs (a uniquely powerful approach to the challenge of testing end-user experience in client/server environments) TrueLog Explorer allows you to confirm visually whether or not embedded objects, text, graphics, table data, SQL responses and more are actually downloaded and displayed by clients while systems are under load. This allows you to detect a class of errors that other Web-traffic simulation tools are not able to detect: errors that occur only under load that are not detected with standard load test scripts.

Content verifications remain useful after system deployment as they can be employed in on-going performance management.

Inserting an Oracle Forms Verification Function

Perform an Oracle Forms Try Script test run. Open the resulting TrueLog in TrueLog Explorer.

1. With an Oracle Forms TrueLog open in TrueLog Explorer, select a TrueLog node that offers verifiable data.
2. Right-click a verifiable value and choose **Verify Value**. The **Add Forms Control Verifications** dialog box opens.
3. Click the **Verify the Value** link. The **Insert Value Verification Function** dialog box appears.
4. Select one of the following values from the drop list:
   - `is equal to`
   - `is different from`
   - `contains`
   - `does not contain`
5. Specify verification case sensitivity and white space details.
6. Specify the severity that is to be raised if the verification returns a negative result.
7. Specify if verification should be against a parameter.
8. Click **OK** on the **Insert Value Verification Function** dialog box. The verification function is added to your test script.
9. Click **Yes** on the **Add Verifications** dialog box to perform a Try Script run.
10. Confirm that verifications have been passed successfully.

    API nodes that include verifications are indicated with blue “V” symbols

Completing Your Oracle Forms Script Customizations

Once the following steps have been completed your Oracle Forms test script should run without error.

- Customize how the script handles session information and user-input data.
- Add any required verification functions.
- Complete any required manual BDL script editing via Silk Performer.

Working With Citrix Applications

In addition to offering TrueLog recording, TrueLog replay, and basic TrueLog analysis for the testing of applications that are hosted by Citrix servers, TrueLog Explorer also offers user-input data customization. The functionality described here can also be applied to the testing of Citrix NFuse sessions.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

Note: For detailed information regarding the use of Silk Performer in testing applications that are hosted by Citrix servers, refer to the *Citrix Tutorial*.

Silk Performer Citrix Player

Silk Performer relies on its own Citrix player for replay, rather than TrueLog Explorer. The Silk Performer Citrix player opens when Try Script runs begin, replaying all recorded actions in full animation. Mouse movements and operations are simulated with an animated mouse icon.

The Silk Performer Citrix player includes a **Log** window with three panes that detail different aspects of each Try Script run:

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 that are accessed during 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/destruction.

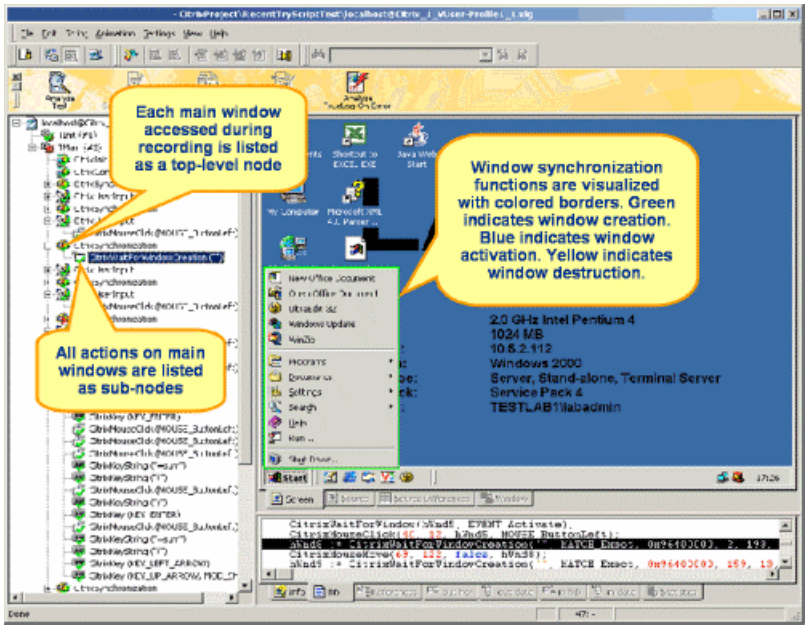
Citrix TrueLogs

TrueLog Explorer may optionally be opened along with the Citrix player during Try Script runs (by selecting the **Animated Run with TrueLog Explorer** check box on the **Try Script** dialog box). TrueLog Explorer displays the data that is actually downloaded during Try Script runs. Each main window accessed during recording is listed as a top-level API node in the TrueLog Explorer menu tree. Actions recorded against those windows are listed as subnodes.

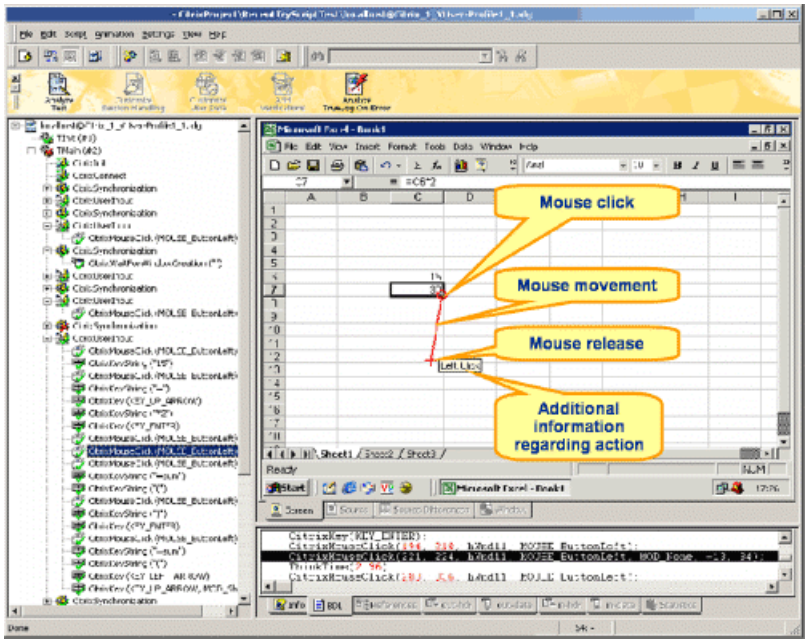
By selecting a high-level synchronization node you see a window as it appeared after the last synchronization function (the bitmap captured during replay).

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.

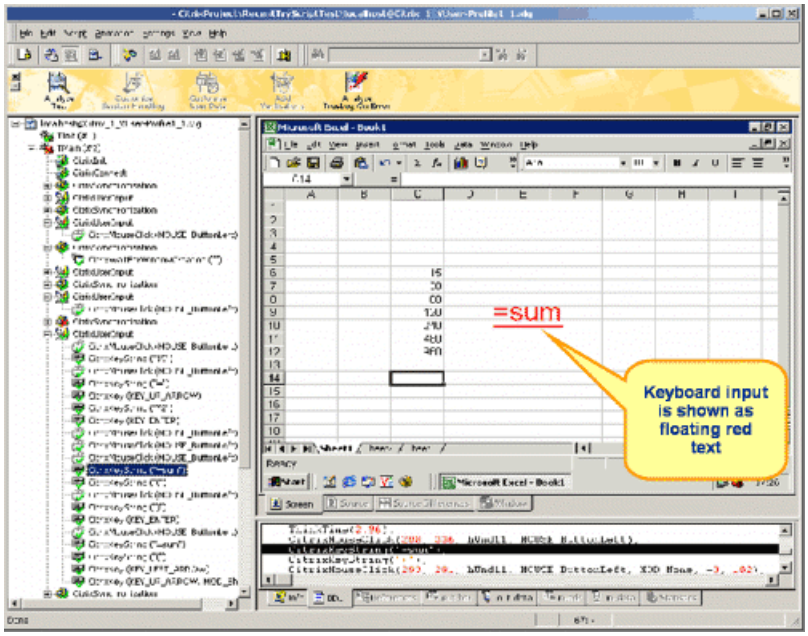
TrueLogs work in complement with the Citrix player by visualizing screen states. For example, if you are not sure which window is indicated by a certain window ID that is listed in the Citrix Player **Log** window, you can find the corresponding synchronization function in the corresponding TrueLog and thereby access a bitmap that shows the window in question.



User input nodes (`CitrixUserInput` and related functions) reflect keyboard and mouse input. `CitrixMouseClick` functions offer two track vector parameters (X and Y coordinates). Red squares 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 (right-click, left-click, double-click, etc).



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.



Synchronization Problems in Citrix Scripts

Windows may fail to be activated and screen synchronizations may fail when the Citrix player 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. Such differences are best assessed visually using a bitmap viewing program.

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. Sometimes there is no user fault and a window is activated only sporadically. In such cases, you must remove the associated `CitrixWaitForWindow` function.

TrueLog Explorer captures screenshots when errors occur (the default setting) and writes these bitmaps to disk. Visual comparison of record and replay screens can be achieved by comparing TrueLog On Error bitmaps with the bitmaps that are captured by Silk Performer along with each synchronization function (the default setting). By default the recorder writes screenshots to the screenshots directory in the project directory. Replay stores screenshots in the current result directory.

Note: The Silk Performer **Dump window region of unsuccessful screen synchronizations** Citrix option must be activated (the default) to have these bitmaps captured and saved.

Citrix User-Input Data Customization

With user-input data customization you can make your test scripts more realistic by replacing static recorded input data with dynamic, parameterized data that changes with each transaction. Manual scripting is not required for 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 by right-clicking within rendered screenshots that are captured during recording.

Customizing Citrix User-Input Data

Use the **Parameter Wizard** to make your test scripts more realistic by replacing static recorded input data with dynamic, parameterized data.

1. Select a node in the menu tree that reflects user data input (for example, select a `CitrixKeyString` node that specifies a keyboard data string).
2. Right-click the input data string (shown as floating red text) and choose **Customize User Input**.
3. The **Parameter Wizard** opens. Select **Create new parameter** and click **Next**.

   With the Parameter Wizard, you can modify script values in either of the following 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.
4. The **Create New Parameter** dialog box opens. Select the **Parameter from Random Variable** option button and click **Next**.
5. The **Choose the kind of usage** dialog box opens. Specify whether the new random value should be used `Per usage`, `Per transaction`, or `Per test`.
6. 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 displays below in **BDL** view.
7. Initiate a Try Script run with the random variable function in your test script to confirm that the script runs without error.

Customizing Mouse Events

The behavior of recorded mouse events can be visually customized.

1. In the menu tree, select a `CitrixMouseClick` node that includes mouse activity. Red squares 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 (right-click, left-click, double-click, etc).

2. Right-click anywhere on the screen and select **Customize User Input**. The **Customize Mouse Event** dialog box opens.
3. Click at the screen position where you want the customized mouse move to begin.
4. Click **Customize** 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.

Citrix Parsing and Verification Functions

Silk Performer support for optical character recognition (OCR) simplifies session-dependent verifications and parsing by recognizing text values in the screenshots of captured application states. As is the case with other TrueLog formats (Web, database, etc), verification and parsing functions are added via TrueLog Explorer after script recording.

Note: To enable optical character recognition (OCR) for parsing and verification functions in TrueLog Explorer, you must first generate a font database using Silk Performer system settings.

Window Position and State

Window positions and state (maximized/minimized) are important for ensuring accurate replay as TrueLog Explorer scripts screen coordinates where selected text is to be read 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 locate the specified text. If it is not possible to specify an absolute position of a conversion-region, your script must be manually updated using coordinates that are relative to windows.

Verification Functions

Although input value verification and response data verification are not supported for Citrix, Silk Performer does support bitmap and window verification for applications that are hosted by Citrix servers.

Inserting OCR Verification Functions

Bitmap and window verification for applications hosted by Citrix servers does not allow for the verification of session-dependent data (for example, login names). However, using OCR techniques, Silk Performer enables the storing of recognizable text values in variables, thereby simplifying session-dependent verifications in Citrix load tests.

1. Using Silk Performer, record a Citrix session.
2. Run a Try Script run, with the **Animated Run with TrueLog Explorer** check box checked on the **Try Script** dialog box. This opens TrueLog Explorer.
3. When the Try Script run is complete, select the API node that includes the bitmap screengrab of the screen on which you want to verify text.
4. Click and drag your cursor onscreen to select the screen region that includes the text you want to use for verification.
5. Right-click in the selected area and choose **Verify Text**.
6. The **Insert Text Verification Function** dialog box opens. The selected text is pre-loaded into the **constant value** text box and the **constant value** option button is selected by default.

   Tip: In addition to verifying against a constant value, you can also verify against a parameter (either an existing parameter or a new parameter). To verify against a parameter, select the **parameter** option button. If a parameter already exists, clicking **[...]** allows you to browse to and select a parameter. If no parameters exist, clicking **[...]** launches the **Parameter Wizard**, enabling you to create a new parameter.

7. Specify the OCR technique that should be used to verify the text. While font-based OCR needs fewer resources, the fuzzy OCR produces better results for texts displayed with font smoothing techniques. Help the fuzzy OCR to provide better results by specifying the language of the text.
8. From the **Verify that the text in the selected rectangle is** list box, select `equal` or `not equal`.
9. Specify whether or not the verification is to be `Case sensitive` and if it should `Ignore whitespaces`.
10. 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`). Click **OK**.
11. A confirmation dialog box opens. Click **OK** to add the OCR verification function to your Citrix test script.

Inserting OCR Parsing Functions

1. Using Silk Performer, record a Citrix session.
2. Run a Try Script run, with the **Animated Run with TrueLog Explorer** check box selected on the **Try Script** dialog box. This opens TrueLog Explorer.
3. When the Try Script run is complete, select the API node that includes the bitmap screengrab of the screen from which you want to parse text.
4. Click and drag your cursor onscreen to select the screen region that includes the text you want to parse.
5. Right-click in the selected area and select **Parse Text**.
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 Silk Performer's **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-time measurements, these files should only be used for debugging purposes and should not be generated for full load tests.
OCR Technique	Specify the OCR technique that should be used to verify the text. While font-based OCR needs fewer resources, the fuzzy OCR produces better results for texts displayed with font smoothing techniques. Help the fuzzy OCR to provide better results by specifying the language of the text.

7. Click **OK**.
8. A confirmation dialog box opens. Click **OK** to add the OCR parsing function to your Citrix script.

OCR Verification and Parsing

This section explains how to parse and verify strings that are generated through OCR.

How OCR Verification and Parsing Works

String verification through optical character recognition (OCR) is achieved using `CitrixVerifyText` or `CitrixVerifyTextFuzzy` API calls. These functions are inserted via TrueLog Explorer during script customization. `CitrixVerifyText` and `CitrixVerifyTextFuzzy` functions compare text strings in replay bitmaps to determine if they are identical.

`CitrixParseText` and `CitrixParseTextFuzzy` functions are available for parsing text.

Silk Performer offers two distinct OCR techniques:

1. Font-based OCR approach: The font-based approach compares text bitmaps to bitmaps of system fonts. This approach relies on pattern databases to recognize varying fonts and text styles. Font databases must be generated before font-based OCR can be used. Characters are only recognized if they can be exactly reproduced by a system font and its styles. OCR results may be poor if font smoothing techniques such as anti-aliasing or ClearType are used. The font-based OCR approach is optimized for performance and resource consumption.
2. Fuzzy OCR with language affinity: Through fuzzy text recognition this approach analyzes text images using a character and language knowledge base. Rather than comparing images to font bitmaps, the fuzzy OCR uses proprietary text and language recognition algorithms.

Only Citrix TrueLogs show verification and parsing API calls in the menu tree. With other TrueLog modes (for example, Web and database), new API nodes are not added to the menu tree.

Note: OCR operations should be performed on static content. When synchronizing on window events it is possible that screen refresh may be slightly delayed, which will result in timing-dependent outcomes. If a screen changes too frequently the correct image may not be selected for comparison if timing is off. Therefore you should script a wait or a `CitrixWaitForScreen` function call before all OCR verification or parsing functions.

Configuring Silk Performer for OCR

To enable optical character recognition (OCR) for Citrix parsing and verification functions using the font-based OCR approach, you must generate a font database via Silk Performer system settings.

Font-based 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 wish 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.

1. In Silk Performer, go to **Settings** > **System...** and choose the **Citrix** icon.
2. On the **Font-based OCR** page, use **Add >>** and **Add All** to move those fonts that you want to have used for OCR from the **System Fonts** list box to the **Chosen Fonts** list box.
3. Use **Remove All** and **<< Remove** to delete unnecessary fonts from the **Chosen Fonts** list box.
4. In the **Sizes** text box, specify the font size range that should be used (for example, `8-20`).
5. Define which font styles should be included by checking the **Italic**, **Bold**, and **Underlined** check boxes.
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.
8. Click the **Fuzzy OCR** tab and select a language. This setting helps the fuzzy OCR engine to deliver better results for a particular language.
9. Click **OK** on the **System Settings** dialog box to accept the changes.

Generating a Text-Synchronization Function

TrueLog Explorer offers the `CitrixWaitForText` synchronization function that pauses script execution until specified text or a text pattern appears at a specified screen position. This feature can be used to synchronize screens and user input. For example, you might want a script to pause execution until a specified dialog text says "ready" instead of "processing" before pressing **OK**. Such an action is difficult to synchronize when you are not sure of processing duration. The scripting of this function would read `CitrixWaitForText(... "ready", ...)`, preceding a `CitrixMouseClick(...)` function.

1. Using Silk Performer, record a Citrix session.

2. Run a Try Script run with the **Animated Run with TrueLog Explorer** check box selected on the **Try Script** dialog box. TrueLog Explorer launches.
3. When the Try Script run is complete, select the API node that includes the bitmap screengrab of the screen on which you want to wait for a specific text string.
4. Click and drag your cursor onscreen to select the screen region that includes the text you want to synchronize.
5. Right-click in the selected area and choose **Synchronize Text**.
6. The **Insert Text Synchronization Function** dialog box opens. Enter a constant value for which the script should wait, or select a parameter value. Click **OK** to insert the function into your script.

Working With TCP/IP and UDP-Based Applications

This section explains how to apply TrueLog On Error and test-script analysis features to the testing of TCP/IP and UDP protocol based applications.

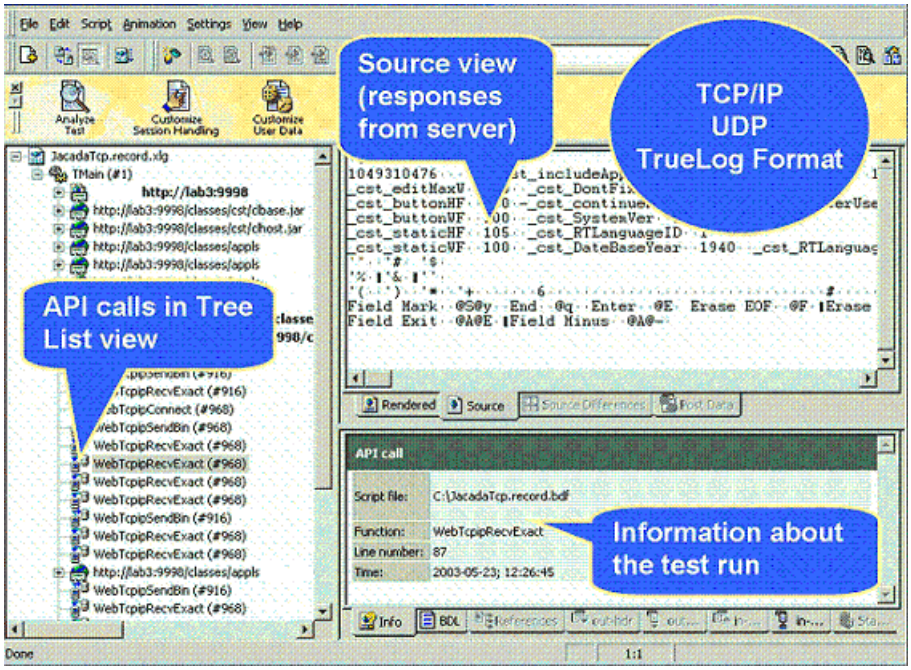
TrueLog Explorer serves as an effective viewer for TCP/IP and UDP based TrueLogs (For example, SMTP, POP3, and custom protocols). TrueLog Explorer enables you to compare replay TrueLogs with record TrueLogs. Script customizations (session handling, user-input data parameterization, and verification functions) are not available for TCP/IP and UDP TrueLogs.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

TCP/IP and UDP TrueLog Structure

TCP/IP and UDP TrueLogs are structured and viewed in the same way that HTML TrueLogs are structured and viewed. One exception is that TCP/IP and UDP do not offer a high-level rendered interface as HTML does.

Display of the **Rendered** page is suppressed for TCP/IP and UDP TrueLogs, except when Web API calls are included in them.



Content Pane

For TCP/IP and UDP based TrueLogs, TrueLog Explorer provides the following data views in the **Content** pane:

Item	Description
Rendered	Suppressed for TCP/IP and UDP, except when Web API calls are included.
Source	Contains the responses sent from the server to the application (for example, `WebTcpipRecv` functions).
Source Differences	The **Source Differences** page lists the in-data (source) differences between selected nodes. This page is helpful in finding dynamic information in server responses. It is the first place to look for session information.  **Note:** The **Source Differences** page is only available in *difference mode.*
Post Data	Suppressed for TCP/IP and UDP.

Information Pane

The **Information** pane offers data regarding TCP/IP and UDP test scripts and test runs. It offers the following views:

Item	Description
Info	The **Info** page displays general information about the open TrueLog file and the selected API node, including: • Script file name • Function • Line number • Time • Duration • Absolute URL • Completion status • Additional information, if available
BDL	The **BDL** page displays the BDL script that corresponds to the open TrueLog. The BDL script is automatically positioned to the line of the selected API node.
References	Suppressed for TCP/IP and UDP
out-hdr	Suppressed for TCP/IP and UDP
in-hdr	Suppressed for TCP/IP and UDP
out-data	For TCP/IP and UDP, this page contains data sent with `WebTcpipSend` functions from the application to the server.
in-data	Contains the responses that are sent from the server to the application (for example, `WebTcpipRecv` functions)
Statistics	Suppressed for TCP/IP and UDP

Setting ASCII and Hexadecimal Viewing Options

Though **Auto view mode** automatically selects the most appropriate data representation mode for you (for example, `binary` for images, `text` for HTML documents), TrueLog Explorer gives you the option of changing your view of TCP/IP and UDP functions between ASCII and hexadecimal format.

1. Choose **Settings** > **Options** > **Display** .
2. Uncheck the **Auto view mode** check box in the **Data format** section of the dialog box.
3. Check the **Show binary data** check box.
4. Click **OK**.

Comparing TCP/IP and UDP Record and Replay TrueLogs

When a TCP/IP and UDP replay TrueLog is compared alongside a corresponding record TrueLog and the dynamics of the protocols are uncovered, those dynamics must be customized for future test runs otherwise session-relevant data will be passed along during tests and errors will result. To customize the dynamics of these protocols, TrueLog Explorer offers TCP/IP and UDP recording rules.

Note: Automatic session handling customization is not available for TCP/IP and UDP based applications. Dynamic BDL parsing functions must be coded manually via Silk Performer.

1. In compare mode, open a TCP/IP or UDP replay TrueLog and the corresponding record TrueLog.
2. Click the **Source** tab.

   Note: Difference tables are not enabled for binary content, so the **Source Differences** page is not available.
3. Click **Step through TrueLog**. The **Step Through TrueLog** dialog box opens.
4. Click the **API calls** option button. The TrueLog API calls with the recorded responses are displayed alongside the corresponding replay responses. Those differences that indicate that dynamics are included in the test script must be customized via Silk Performer recording rules.

Working With Terminal-Emulation Applications

This section explains how to customize a terminal emulation test script based on the results of a Try Script run.

Working With Terminal-Emulation Applications - Overview

In addition to offering TrueLog recording, TrueLog replay, and basic TrueLog analysis for terminal-emulation applications, TrueLog Explorer also offers parsing, content verification, and input-parameter customization. TrueLog functionality is supported for the following protocols: `VT100+`, `IBM 3270`, and `IBM 5250`.

Before proceeding with this chapter, it is essential that you familiarize yourself with TrueLog Explorer basic functionality.

User Data Customization

User data customization via TrueLog Explorer and the Parameter Wizard is supported for both TN3270 and TN5250 protocol terminal emulation applications.

On-Screen Event Synchronization

Functions are available for synchronization of on-screen events in record and replay scenarios, though the functions cannot be inserted using TrueLog Explorer.

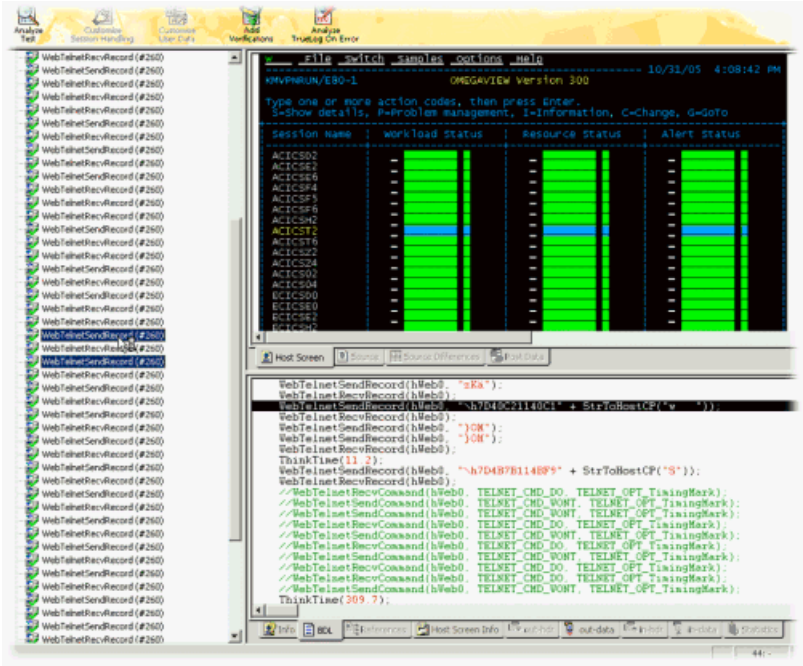
Note: For additional details regarding Silk Performer support for terminal emulation applications, please see `Miscellaneous Tutorials`, a PDF that accompanies your Silk Performer installation at **Documentation** > **Tutorials**.

Terminal-Emulation TrueLog Structure

TrueLog Explorer supports the visualization of terminal emulation requests and responses in the same way that it supports the visualization of HTTP client requests and HTTP/HTML server responses.

The three windows that are displayed with terminal emulation TrueLogs are listed below:

- **TrueLog menu tree** – Lists all terminal emulation API calls that were included in the test run.
- **Content pane** – Displays the state of the terminal emulation application at each API node.
- **Information pane** – Displays data related to the most recent test run. The view tabs in this pane that are active and applicable to terminal emulation TrueLogs are **Info**, **BDL**, **in-data**, **out-data**, and **Host Screen Info**.



Customizing Host Screen Display

You can change the font or font size of text displayed in **Host Screen** view (within the Content pane). This may be necessary if the pre-defined font (lucida console) does not correctly display the characters that are received by the application under test.

Font type:

To change the font, you must change the following key in the Windows registry:

```
HKEY_CURRENT_USER\Software\Silk\Silk Performer\16.0\SystemSettings\Layout
\GreenScreen Font Name
```

Font size:

To change the size of the displayed characters, you must change the following key in the Windows registry:

```
HKEY_CURRENT_USER\Software\Silk\Silk Performer\16.0\SystemSettings\Layout
\GreenScreen Font Size
```

Enter the size of the font (default is *10*).

Stepping Through Terminal-Emulation TrueLogs

After tests, it is typical to have multiple TrueLog On Error files open in TrueLog Explorer (one TrueLog for each virtual user who returns an error). With TrueLog Explorer Find Errors feature, you can jump from one error to the next sequentially as they occurred in time, regardless of which TrueLogs the errors were recorded in. This simplifies the process of analyzing errors. There is no need for you to manually review all open TrueLogs to find the next error in a sequence.

1. After the completion of a test, within Silk Performer, click **Explore Results**.

   Note: TrueLog On Error files are generated only when the Silk Performer **Generate TrueLog On Error** option is enabled.

   The **Explore Results** dialog box opens.
2. Click TrueLog Explorer.

   Note: TrueLog Explorer is disabled both when no errors are detected during a test and when Truelog On Error is not enabled. In such instances, proceed directly to analyzing results with Performance Explorer.

   TrueLog Explorer opens with the TrueLog On Error files that were generated for the current test, and the **Step through TrueLog** dialog box opens.
3. Using the **Step through TrueLog** dialog box, click the appropriate option button to advance through all `Stable Screens` (`WebTelnetRecvCommand` calls), `API calls` (each API node), and `Errors`.

   TrueLog On Error files are searched sequentially as they were recorded in time.

   Note: When multiple `WebTelnetRecvCommand` calls are clustered in a sequence, clicking **Find Next** with the `Stable Screens` option automatically advances focus to the final `WebTelnetRecvCommand` call in the sequence.
4. Click **Find Next** to advance to the first stable screen, API call, or error.

   Error messages are displayed on the **Info** page.

   API nodes that contain replay errors are tagged with red “X” marks in the menu tree.

To view the originally recorded TrueLog that corresponds to the open replay TrueLog, right-click the replay TrueLog’s virtual-user node in the menu tree and choose **Open Associated Record TrueLog**. Each server response is then available for viewing on the **Host Screen** page.

Analyzing Terminal-Emulation Test Scripts

Once you have generated a test script you should determine if the script runs without error by performing a Try Script run. A Try Script run will determine if the script accurately recreates the recorded interactions.

The default option settings for Try Script runs include live display of data downloaded during the test run and the writing of log and report files.

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 delay between transactions.

1. Click **Try Script** on the workflow bar. The **Try Script** dialog box opens.
2. To view rendered page transitions during a Try Script run, check the **Animated Run with TrueLog Explorer** check box.

3. Click **Run**.

   Note: You are not running an actual load test here, only a test run 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.

   TrueLog Explorer opens, showing you the data that is actually downloaded during the Try Script run. Each main terminal emulation window accessed during recording is listed as a high-level `WebTelnetRecvCommand` API node in the menu tree.

Comparing Replay and Record TrueLogs

When testing terminal emulation applications, differences can occur in corresponding field values between replay and record TrueLogs. Some differences may result in replay errors.

1. Open a replay TrueLog and click **Analyze Test** on the workflow bar. The **Analyze Test** dialog box opens.
2. Click **Compare your test run**.
3. The associated record TrueLog opens in **compare view**. The **Step through TrueLog** dialog box also opens.
4. Select an interval to search by (`Stable Screens`, `API nodes`, or `Errors`).
5. Click **Find Next** to advance to the first set of corresponding stable screens, API calls, or errors in the record and replay TrueLogs. Visually compare the values and states of the controls and screenshots to see if there are any differences.
6. Once you have completed any required customizations (based on differences you have discovered), click **Try Script Run** to confirm that your customizations run without error.
7. Analyze the results of subsequent test runs to determine whether your customizations were successful or if further customization is required.
8. Click **Compare Mode** to disable compare mode.
9. Repeat this procedure as many times as is required until your script is fully customized and all necessary customizations have been added.

Synchronizing Record and Replay TrueLogs

When in compare mode, you can synchronize corresponding API nodes between replay and record TrueLogs.

1. Enable compare mode.
2. Open a set of record and replay TrueLogs.
3. Right-click an API node for which you want to establish synchronization.
4. Select **Synchronize TrueLogs**. TrueLog Explorer then locates the API node in the matching TrueLog that best correlates with the selected API node.

Terminal-Emulation TrueLog Functions

Six terminal emulation functions are available for insertion into test scripts.

WebTelnetScreenVerifyText	Verifies onscreen text (available from **Host Screen** view).
WebTelnetScreenGetText	Parses onscreen text (available from **Host Screen** view).
WebTelnetScreenVerifyField	Verifies a text field (available from **Host Screen Info** view).
WebTelnetScreenGetField	Parses text from a field (available from **Host Screen Info** view).

WebTelnetScreenVerifyStatus	Verifies a field's status (available from both views).
WebTelnetScreenGetStatus	Parses a field's status (available from both views).

Customizing User Input Data

Note: Visual user input data customization for terminal emulation is supported for both TN3270 and TN5250 protocol scripts.

1. With a terminal emulation TrueLog loaded into TrueLog Explorer, select the `WebTelnetSendRecordExecute` API call (in the **API node tree menu**) that includes the user input data that is to be customized.
2. Select the **Host Screen** tab.

   The **Host Screen** tab displays a screen capture of the user input screen.
3. Select the **Host Screen Info** tab.

   The **Host Screen Info** tab includes a list of all user input data fields that are available on the selected screen. The input fields are numbered and listed in the order in which they appear on the screen. When you select an input field entry on the **Host Screen Info** tab the corresponding field is highlighted in the screen capture on the **Host Screen** tab.

   Note: Only input fields that had data entered into them during recording are highlighted and available for customization.
4. On the **Host Screen Info** tab, right-click the field entry that you want to customize and select **Customize User Input**. The **Parameter Wizard** launches.
5. Use the **Parameter Wizard** to define how you want input data parameters to appear in your script.

   See Silk Performer Help for details on working with the **Parameter Wizard**.

   Modifications that you make to your script using the **Parameter Wizard** can be viewed on the BDL tab.

Execute a Try Script run to verify that the new parameter inserts user input data as required.

Verification Functions for Terminal-Emulation Applications

This section explains how to verify content and fields for terminal emulation API calls.

Verification Functions for Terminal-Emulation Applications

TrueLog Explorer allows you to easily add content verifications to test scripts to verify that the content that is to be sent by servers is in fact received by clients under real-world conditions. Content verification functions may be inserted for any terminal emulation API call. For example, for a `WebTelnetSendCommand` function call in which input data is entered, you can insert a return value verification function. Verification functions can be inserted through context menus in both **Host Screen** view and **Host Screen Info** view.

By comparing replay test runs with record test runs, a uniquely powerful approach to the challenge of testing end-user experience in client/server environments, TrueLog Explorer allows you to confirm visually whether or not input data are actually downloaded and displayed by clients while terminal-emulation applications are under heavy load. This allows you to detect a class of errors that other terminal emulation traffic simulation tools are not able to detect: errors that occur only under load that are not detected with standard load test scripts.

Content verifications remain useful after system deployment as they can be employed in ongoing performance management.

By right-clicking an input field that you want to have verified in **Host Screen** or **Host Screen Info** views, the required verification function can be generated and inserted into your BDL script. TrueLog Explorer

offers three pre-enabled verification functions (`WebTelnetScreenVerifyText`, `WebTelnetScreenVerifyField`, and `WebTelnetScreenVerifyStatus`).

Content Verification vs. Field Verification

Content verifications enable you to select and setup verifications for onscreen text strings in **Host Screen** view, regardless of their placement on screen. *Field verifications* enable you to set up verifications for pre-defined regions (fields) within a terminal view. An application might define an entire screen as a single field, or a page might be built of header fields, footer fields, body fields, and so forth. Fields are often used as input areas, but they do not necessarily need to accept input. On-screen characters do not necessarily need to reside within fields.

Inserting a Content-Verification Function

1. In **Host Screen Info** view, right-click an input field for which you want to verify the return value.
2. Select **Verify Selected Text**.
3. The **Insert Verification Function** dialog box opens. Use this dialog box to specify the type of verification function that should be inserted into the BDL script. From the **Verify that the sequence of characters starting at column x, row x** list box, select `is equal to` or `is not equal to`.
4. Specify if the verification should be against a `constant value` or a `parameter value`.
5. Specify whether or not the verification is to be `Case sensitive` and if white spaces should be ignored.
6. 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`).
7. Click **OK** to add the verification function to your test script. Once the BDL script has been successfully modified, repeat this process for each verification you want to add to the BDL script.

Inserting a Status-Verification Function

A status-verification function enables you to verify a status value in a terminal-emulation input field. A *status* is a name-value pair (it is the value of the status name that is verified).

1. Right-click an input field in **Host Screen Info** view.
2. Choose **Verify Status Value**.
3. The **Insert Status Value Verification Function** dialog box opens. Use this dialog box to specify the type of verification function that should be inserted into the BDL script. From the **Verify that the value of the status** list box, select the name of the status you want to verify.
4. Select `is equal to` or `is not equal to` to define the status state you want to verify.
5. Specify if the verification should be against a `constant value` or a `parameter value`.
6. Specify whether or not the verification is to be `Case sensitive` and if white spaces should be ignored.
7. 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`).
8. Click **OK** to add the status-verification function to your test script. Once the BDL script has been successfully modified, repeat this process for each status verification that you wish to add to the BDL script.

Inserting a Field-Verification Function

You can define a field-verification function for a terminal-emulation input field to verify text in the field.

1. Right-click a field in **Host Screen Info** view and choose **Verify field**.
2. The **Insert Field Verification Function** dialog box opens. Use this dialog box to specify the type of verification function that should be inserted into the BDL script.

3. Select `is equal to` (or `is not equal to`) as the operator for the verification.
4. Specify if the verification should be against a `constant value` or a `parameter value`.
5. Specify whether or not the verification is to be `Case sensitive` and if white spaces should be ignored.
6. 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`).
7. Click **OK** to add the verification function to your test script. Once the BDL script has been successfully modified, repeat this process for each additional verification that you want to add to your BDL script.

Parsing Functions for Terminal-Emulation Applications

This section explains how value parsing can be used to enable accurate replay when session-specific strings must be parameterized.

Parsing Functions for Terminal-Emulation Applications

A parsing function can be inserted for any screen text, field, or status in a terminal-emulation TrueLog. Value parsing can be useful for enabling replay when session-specific strings must be parameterized to enable accurate replay (for example, you might parse newly created customer IDs into parameters that are inserted into the customer ID fields of invoices).

Content Parsing vs. Field Parsing

Content parsing functions enable you to select and set up parsing for onscreen text strings in **Host Screen** view, regardless of their placement on screen. *Field parsing functions* enable you to set up parsing for pre-defined regions (fields) within a terminal view. An application may define an entire screen as a single field, or a page may be built of header fields, footer fields, or body fields. Fields are often used as input areas, but they do not necessarily need to accept input. Onscreen characters do not necessarily need to reside within fields.

Inserting a Content-Parsing Function

1. Within **Host Screen** view, select text for which you want to parse a value.
2. Right-click and choose **Parse Selected Text**.
3. The **Insert Parsing Function** dialog box offers settings by which the parsing function can be adjusted.

   Though the default settings will likely be appropriate, you can adjust the following settings:

   - **Parameter name**: Enter a name for the parameter that is to receive the result of the parsing function.
   - **Informational statement insertion**: Check **Print statement** to insert an informational `Print` statement into the script. This will write the result of the parsing function to the Silk Performer **Virtual User Output** window. Check **WriteIn 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 the time measurements of tests, these files should only be used for debugging purposes and should not be generated for full load tests.
4. Click **OK** to insert the parsing statement into your test script.

Inserting a Status-Parsing Function

1. Right-click a status in **Host Screen Info** view and choose **Parse Status Value**.

   Note: This function can also be accessed by right-clicking a field in **Host Screen** view.
2. The **Insert Status Parsing Function** dialog box offers settings by which the parsing function can be adjusted.

Though the default settings will likely be appropriate, you can adjust the following settings:

- **Status**: From the list box of existing status names, select the status for which you want to insert a parsing function.
- **Parameter name**: Enter a name for the parameter that is to receive the result of the parsing function.
- **Informational statement insertion**: Check **Print statement** to insert an informational `Print` statement into the script. This will write the result of the parsing function to the Silk Performer **Virtual User Output** window. Check **WriteIn 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 the time measurements of tests, these files should only be used for debugging purposes and should not be generated for full load tests.

3. Click **OK** to insert the parsing statement into your test script.

Inserting a Field-Parsing Function

1. Right-click a field in **Host Screen Info** view and choose **Parse Field Text**.
2. The **Insert Field Parsing Function** dialog box offers settings by which the parsing function can be adjusted. Though the default settings will likely be appropriate, you can adjust the following settings:

   - **Parameter name**: Enter a name for the parameter that is to receive the result of the parsing function.
   - **Informational statement insertion**: Check **Print statement** to insert an informational `Print` statement into the script. This will write the result of the parsing function to the Silk Performer **Virtual User Output** window. Check **WriteIn 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 the time measurements of tests, these files should only be used for debugging purposes and should not be generated for full load tests.

3. Click **OK** to insert the parsing statement into your test script.

Working With AJAX-Enhanced Web Applications

This section explains how to "pretty" format JSON and XML data and how to apply TrueLog Explorer script customization features to the testing of applications that utilize AJAX techniques.

The Silk Performer recorder can record and (with the assistance of TrueLog Explorer) 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.

For more information about AJAX, see the Silk Performer Help.

AJAX-Support Overview

TrueLog Explorer and Silk Performer enable access to values within AJAX requests. This facilitates TrueLog Explorer script customizations such as input-data parameterization, verification functions, parsing functions, and session-information customization within AJAX responses.

Pretty Formatted JSON and XML Data

JSON and XML are data-structure formats commonly used in AJAX applications, REST techniques, and other environments. TrueLog Explorer supports pretty-formatted viewing of XML and JSON-formatted byte streams in recorded scripts. Enhanced rendering of JSON formatted data enables the customization of string values via TrueLog Explorer customization functions, such as string verification and parsing.

Pretty-formatted JSON view is also supported in compare mode, in which you can compare record and replay sessions side by side and automatically detect differences.

Note: TrueLog Explorer offers the option of viewing JSON data in either pretty-formatted JSON-rendering view or as a raw JSON byte stream, though script customization features such as verification and parsing are not supported in raw data view.

Pretty-formatted data viewing is also available in Silk Performer to facilitate readability and B.F. script customizations.

Enabling Pretty-Formatted JSON and XML Viewing in TrueLog Explorer

Enable pretty-formatted JSON and XML viewing on the **Rendered** 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 **Rendered** 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 **Rendered** tab.
3. Click **Auto View** on the toolbar. JSON and XML data is automatically pretty-formatted.
4. Right-click JSON or XML data and choose **Render As** > **JSON** or **Render As** > **XML** to pretty-format the data.

Customizing TrueLog Explorer

This section explains how to adapt TrueLog Explorer customization, view, and TrueLog generation features to your specific needs. TrueLog Explorer option settings enable you to configure the display of data, the definition of delimiters for detecting differences, and the generation of virtual user reports. Customization features enable you to customize UI commands and the display of toolbars.

Compare mode enables the direct comparison of record and replay TrueLogs, one API node at a time. Differences mode automatically identifies and lists differences between record and replay TrueLogs for Web, XML, and database applications via the **Source Differences** page. Differences mode also displays differences in text-based format on the **Source**, **in-data**, **out-data**, **in-hdr**, **out-hdr**, and **BDL** pages by rendering differing text in green and red.

When enabled, TrueLog Explorer animation displays replay content (graphics, text, and SQL commands) in real-time as it is downloaded.

Finally, as the generation of TrueLog files can be CPU/memory intensive, this section shows you how to use Silk Performer to adjust the characteristics of TrueLogs and the criteria by which they are generated.

TrueLog Explorer Option Settings

TrueLog Explorer option settings, available on the **Options** dialog box, enable you to configure application settings such as the display of HTML documents, incoming and outgoing data, the definition of delimiters (which pinpoint differences between TrueLogs), and the generation of virtual user reports.

Setting Display Options

1. Choose **Settings** > **Options** . The **Display** page opens.
2. Specify how HTML documents are rendered.
   - To enable the execution of client-side scripts when rendering HTML documents, check the **Script execution** check box.

- To view images or frames when rendering HTML documents, check the **Frame and image loading** check box.
- To enable the execution of applets or ActiveX controls when rendering HTML documents, check the **Applet execution** check box.

3. Specify how incoming and outgoing data should be displayed on the **out-hdr**, **out-data**, **in-hdr**, and **in-data** pages.
   - To have TrueLog Explorer automatically select the best representation view for data (such as `binary` for images and `text` for HTML documents), check the **Auto view mode** check box.
   - To view text-formatting symbols, such as `\r\n` for a carriage return and new line and "`.`" for a white space, check the **Show white spaces** check box.

     This option is only relevant when viewing data in text format.
   - To view data in binary format, check the **Show binary data** check box.

     This option is only available when **Auto view** mode is not enabled.
4. To use a filter for the TrueLog menu tree, check the **Apply filter** check box. For example, you might use a filter to prevent the TrueLog menu tree from showing certain types of files (images, scripts, or style sheets).
5. Click **OK**.

Inserting Delimiter Characters

TrueLog Explorer general compare tags allow you to define delimiters, which are characters or text that delimit differences between TrueLogs. Delimiters determine if a sequence is handled as a single difference or as multiple differences. Delimiters are used for all document types except HTML.

1. Choose **Settings** > **Options** and then click the **Compare tags** tab. The **Compare tags** page opens.
2. In the **General compare tags** table, type character or text delimiters.

   Tip: Click **Default** to discard your changes and revert to the default settings.
3. Click **OK**.

Inserting HTML Delimiter Characters

HTML delimiters are characters or text that delimit differences between HTML TrueLogs. They determine if a sequence is handled as a single difference or as multiple differences.

1. Choose **Settings** > **Options** and then click the **Compare tags HTML** tab. The **Compare tags HTML** page opens.
2. In the **HTML compare tags** table, type character or text delimiters.

   Tip: Click **Default** to discard your changes and revert to the default settings.
3. Click **OK**.

Enabling Virtual-User Summary Reports

The TrueLog Explorer workspace option enables the automatic display of virtual user reports at the end of animated Try Script runs or when the root nodes of TrueLog files are selected in the menu tree.

1. Choose **Settings** > **Options** and then click the **Workspace** tab. The **Workspace** page opens.
2. Check the **Display virtual user report** check box.

   Tip: Click **Default** to discard your changes and revert to the default setting.
3. Click **OK**.

Customizing Toolbars and Commands

This section explains how to customize toolbars and the commands that display on toolbars.

Specifying Which Toolbars are Displayed

You can customize the set of toolbars that appear on the TrueLog Explorer interface.

1. Choose **Settings** > **Customize** . The **Toolbars** page opens.
2. Check the check boxes next to the toolbars that you want to have displayed.
3. *Optional*: To specify that you want to enable tooltips (roll-over UI control descriptions), check the **Show Tooltips** check box.
4. *Optional*: To specify that you want to enable TrueLog Explorer “cool look,” check the **Cool Look** check box. Cool look replaces Windows 3.1 style drop-shadow buttons with non-drop-shadow buttons.
5. Click **OK**.

   Alternative: Click **Apply** to immediately apply your selections to the UI.

Creating a Custom Toolbar

1. Choose **Settings** > **Customize** . The **Toolbars** page opens.
2. Click the **Toolbars** tab.
3. Click **New**. The **New Toolbar** dialog box opens.
4. Enter a name for the toolbar and then click **OK**.
5. Click **OK**.

Customize the command buttons that are displayed on the new toolbar.

Customizing Toolbar Command Buttons

You can define which command buttons display on specific toolbars.

Note: The **Menu** toolbar cannot be customized.

1. Choose **Settings** > **Customize** . The **Toolbars** page opens.
2. Click the **Commands** tab.
3. In the **Categories** list box, select the toolbar for which you want to customize the command buttons.
4. Click available commands in the **Buttons** box to read their descriptions in the **Description** box.
5. Drag the commands that you want to include on the toolbar onto the toolbar name in the **Categories** list box.
6. Click **OK**.

View Modes

You can view differences in TrueLogs using compare and difference modes. Compare mode enables the direct comparison of record and replay TrueLogs, one API node at a time. Differences mode automatically identifies and lists differences between record and replay TrueLogs for Web, XML, and database applications via the **Source Differences** page. Differences mode also displays differences in text-based format on the **Source**, **in-data**, **out-data**, **in-hdr**, **out-hdr**, and **BDL** pages by rendering differing text in green and red.

Compare Mode

In compare mode, replay TrueLogs are compared alongside the corresponding TrueLogs that were generated during application recording. Default view displays replay TrueLogs (with a green triangle in the upper-left corner). Compare view displays record TrueLogs (with a red triangle in the upper-left corner).

TrueLogs are opened in compare view when a corresponding record TrueLog is open or when the **Open in Compare View** check box is checked on the **Open** dialog box.

Note: TrueLogs are automatically opened in compare mode when you execute **Find Differences** from the **Workflow - Customize Session Handling** dialog box.

Enabling Compare Mode

Use compare mode to compare a replay TrueLog alongside a corresponding record TrueLog to analyze playback errors. When you check the **Open in Compare View** check box on the **Open** dialog box, TrueLogs are opened in compare mode automatically. Otherwise, TrueLogs are opened in default view.

Note: Compare mode offers the option of having record and replay **Source** pages oriented vertically rather than horizontally. To compare **Source** pages vertically, choose **View** > **Compare Vertically** .

1. Select a TrueLog in the TrueLog menu tree.
2. Perform one of the following steps:
   - Click **Compare Mode**.
   - Choose **View** > **Compare Mode** .

> **Note:** Multiple entry points and approaches are available for the comparison of TrueLogs.

Difference Mode

Comparing TrueLogs that are generated during Try Script runs alongside the corresponding TrueLogs that were generated during application recording is an effective means of pinpointing problems in load testing scripts.

Difference tables, available from the **Source Differences** page, automatically list the differences that are detected between replay TrueLogs and corresponding record TrueLogs.

Difference mode, which is enabled by default, is a special kind of compare mode in which text differences in the **Source**, **in-data**, **out-data**, **in-hdr**, and **out-hdr** pages are marked in color (red and green with a gray background).

Enabling Difference Mode

Note that difference mode is enabled by default.

Perform one of the following steps:

- Choose **View** > **Diff Mode** .
- Click the **Diff Mode** icon.

Viewing a Difference Table

Ensure that difference mode is enabled before you begin this task (difference mode is enabled by default).

1. Open a pair of corresponding record and replay TrueLogs.
2. Enable compare mode by choosing **View** > **Compare Mode** .

3. Click the **Source Differences** tab. The differences that are detected between replay TrueLogs and corresponding record TrueLogs are listed automatically.

Data Animation

When animation is enabled, content received from servers during virtual user execution is displayed in real time, as users would see it. In the case of Web application testing, this means that you can see page content loading (graphics, text, and embedded objects) as virtual users see it. Animation is enabled by default.

TrueLog animation is helpful when you execute a load test with TrueLog On Error enabled and you detect errors with individual users using the error count in Silk Performer **Virtual User** view in the **Monitor** window. You can view TrueLog On Error files immediately using the context menu within **Virtual User** view by clicking **Explore TrueLog**. However, this only works for virtual users that are running on the controller machine, as you do not have access to TrueLogs for remote agents during load tests.

Pausing, Resuming, and Stopping Animation

To pause or resume animation choose **Animation** > **Pause Animation** . *Alternative*: Press `Ctrl+P` on your keyboard.

To stop animation choose **Animation** > **Abort Animation** .

TrueLog Impact on Scalability

TrueLog generation is only useful in as much as it is adaptable to your load-testing environment. Therefore, memory and performance requirements must be kept to a minimum.

The two major factors affecting TrueLog scalability are:

- Performance downgrade
- Memory usage

Performance Downgrade

Performance downgrades related to TrueLog generation can range from 0% - 10%. This is because runtime keeps all relevant data in memory and attempts to keep necessary memory allocations to a minimum. Note that the performance impact of TrueLog On Error is negligible.

Memory Usage

In terms of scalability, there is a tremendous difference between TrueLog and TrueLog On Error. The TrueLog feature immediately writes each request and response to a TrueLog file. The formatting of the TrueLog file produces high internal CPU usage and a large amount of IO for the writing of the TrueLog file.

As a result, TrueLog generation is often not suitable for load testing.

Since TrueLogs can become large in size, you can configure Silk Performer to generate TrueLogs only when errors are encountered during testing. While systems run accurately, nothing is recorded. Such targeted TrueLog generation is known as *TrueLog On Error* and results in smaller, focused TrueLog files.

Memory requirements for TrueLog On Error are largely dependant on the BDL scripts that are utilized. Therefore, generic percentages can not be given, especially in cases where TrueLog On Error is configured to log complete transactions. Note that this applies only to TrueLog On Error files for which the non-default **Store TrueLog for one transaction** setting has been selected. In such cases, memory usage correlates with the number and size of requested Web pages included in transactions. With the default **Store TrueLog based on content history** setting, the length of transactions has no impact. Content histories usually do not exceed five Web pages, and these are the only pages that must be stored.

Custom Content Types and File Extensions

TrueLog Explorer supports custom handling of special content types and URL file extensions. You can add custom content-types and file extensions to some values in the Registry to change product behavior. The relevant settings can be found in the registry (`HKEY_CURRENT_USER\Software\Silk\Silk TrueLog Explorer\15.5\`).

The following four registry values exist, among others:

- `MaskContentTypes`
- `MaskExtensions`
- `SuppressContentTypes`
- `SuppressExtensions`

These values contain lists of content-types and file extensions that need special handling. The `Mask*` values apply to textual data, which can be rendered as plain text in HTML view. The `Suppress*` values apply to binary (non-text) data, which can not properly be rendered.

Suppressing File Download Dialog Boxes

1. In TrueLog Explorer, select the node where the file download dialog box opens.
2. Click the **in-hdr** tab and examine the page.
3. If the content-type seems unfamiliar, such as `app/fireclick.x-hint.1`, perform the following steps:
   a) Close TrueLog Explorer.
   b) Add the unfamiliar content-type to the `MaskContentTypes` value.

      Strings are separated by semicolons.
   c) Start TrueLog Explorer.

      Registry settings are only read upon startup.
   d) If the download box still opens, add the previous content-type to the `SuppressContentTypes` value. The File Download dialog box should no longer open.
4. If the content-type seems okay, examine the URL.
   a) If you notice an extension other than `.html`, `.js`, `.pl`, `.chtml`, `.jar`, `.class`, or `.vbs`, close TrueLog Explorer and examine the **in-data** page to see whether the response body consists of textual or binary data.
   b) If you see textual data on the **in-data** page, add the URL file extension to the `MaskExtensions` value.
   c) If the data seems to be binary, add the file extension to the `SuppressExtensions` value.

      Extensions are separated by semicolons.
   d) Start TrueLog Explorer.

      Registry settings are only read upon startup.

TrueLog Generation Settings

Overview

TrueLogs are usually not enabled during load testing as they are CPU and I/O intensive. Typically, TrueLogs are enabled during script development/verification only. TrueLog On Error files are usually enabled during load testing as they are optimized for performance and have only a slight impact on CPU. TrueLog On Error files can be activated for large tests without major impact on replay performance, even when a moderate number of errors are anticipated.

Silk Performer's generation of TrueLog On Error has a slight impact on scalability however. To keep this impact as low as possible, the Silk Performer TrueLog recording level can be adjusted in the following ways:

- TrueLog length
- Severity of incidents for which TrueLog is recorded
- Inclusion of embedded objects

The TrueLog generation options detailed in the following sections can be configured by choosing **Settings** > **Active Profile** > **Results** > **TrueLog** within Silk Performer. This same page is also used to enable the generation of TrueLog On Error (`.xlg`) files for each virtual user in tests (using the **TrueLog On Error files (.xlg)** check box).

Length of TrueLogs

TrueLog On Error tracking can be configured to store complete transactions one transaction at a time, or in the case of Web applications, the five most recent pages of content history.

Generating TrueLog based on complete transactions - This setting tracks each erroneous transaction, rather than all relevant pages (content history). As memory consumption is dependent on the length of each transaction, use this option only when memory is of no concern or when transactions are short.

Generating TrueLog based on content history - This setting stores recent page history only, and drops non-relevant pages from memory. This approach makes memory consumption more predictable and manageable. Although, consumption is still dependent on the sizes of pages stored in memory.

Severity

You can define the severity of incidents that trigger the writing of TrueLog data to disk (initially stored in RAM) using the **Generate TrueLog On Error for** setting.

Embedded Objects

Log all embedded objects - TrueLogs contain the response bodies of all embedded objects. Even when request and response headers are no longer contained in the history, the content of embedded objects may still be displayed in the browser. This can be a matter of concern when a page does not request an image from a server because it is a cache hit or when the server responded with a 304 not modified error. To guarantee that all embedded objects are displayed, Silk Performer stores the contents of each embedded object in a global process-spanning cache.

Exclude cached embedded objects - When a global store for cached objects is not enabled, some images will not load. This setting provides a small gain in performance and requires less memory.

Exclude all embedded objects - When embedded object bodies are not logged at all, embedded objects do not load. This setting significantly reduces memory consumption. However, the visual information that is offered with this approach is greatly compromised.

In cases where TrueLog On Error information is recorded based on content history, the impact on memory usage per virtual user can be estimated based on sample tests. The numbers below were calculated based on sample tests run against well known Web sites, including Amazon.com and IBM.com.

Setting	Additional memory consumption
Exclude all embedded objects	80%
Exclude cached embedded objects	125%
Log all embedded objects	130%

Note: The settings that require the least amount of memory (storing TrueLog data based on content history and excluding all embedded objects) are sufficient for logging all relevant data. However, the visual information they offer is greatly compromised.

Parsing and Verification Functions

The matrices in these topics map out the interactions between various parsing and verification functions, the Silk Performer recorder, and TrueLog Explorer.

For full details regarding all available BDL functions, see the BDL Reference.

HTML Parsing and Verification Functions

HTML Parsing Functions

HTML Content	Response Data	Usage	Description
`WebParseHtmlBound`	`WebParseDataBound`	visual parsing, scripting	Parses HTML content/ response data within given boundaries
`WebParseHtmlBoundEx`	`WebParseDataBoundEx`	visual parsing, scripting	Parses HTML content/ response data within given boundaries
`WebParseHtmlBoundArray`	`WebParseDataBoundArray`	scripting	Parses HTML content/ response data within given boundaries, subsequent left boundary searched only after right boundary identified
`WebParseHtmlTitle`	-	scripting	Parses the title of an HTML document
`WebParseTable`	-	visual parsing, scripting	Parses the content of a cell in a HTML table
-	`WebParseResponseTag`	scripting	Parses the value of a specified HTML attribute. Also applicable for XML response data.
-	`WebParseResponseTagContent`	scripting	Parses the content of a specified HTML attribute. Also applicable for XML response data.
-	`WebParseResponseHeader`	scripting	Parses HTTP response headers.
-	`WebParseResponseRedirect`	scripting	Parses the URL of a redirection response for URL encoded parameters.

HTML Verification Functions

HTML Content	Response Data	Usage	Description
`WebVerifyHtml`	`WebVerifyData`	visual verification, scripting	Checks to see if a specified string occurs in HTML content/response data
`WebVerifyHtmlBound`	`WebVerifyDataBound`	visual verification, scripting	Checks to see if a specified string occurs in HTML content/response data within given boundaries
`WebVerifyHtmlBoundEx`	`WebVerifyDataBoundEx`	visual verification, scripting	Checks to see if a specified string occurs in HTML content/response data within given boundaries
`WebVerifyHtmlTitle`	-	visual verification, automatically generated by the recorder, scripting	Checks to see if a specified string occurs in the title of an HTML document
`WebVerifyTable`	-	visual verification, scripting	Checks to see if a specified string occurs in a HTML table
`WebVerifyHtmlDigest`	`WebVerifyDataDigest`	Automatically generated by the recorder, visual verification	Checks if the response data differs from response data that the recorder captured during an earlier recording session

XML Parsing and Verification Functions

XML Parsing Functions

XML Content	Response Data	Usage	Description
-	`WebXmlParseNodeValue`	TrueLog Explorer visual parsing, scripting	Parses a response document and returns the XML node value of a passed XPath
-	`WebXmlParseNodeAttribute`	TrueLog Explorer visual parsing, scripting	Parses a response document and returns the XML attribute value of a node with a passed XPath.

XML Verification Functions

XML Content	Response Data	Usage	Description
-	`WebXmlVerifyNodeValue`	TrueLog Explorer visual verification, scripting	Checks to see if a node's value matches the given value.
-	`WebXmlVerifyNodeAttribute`	TrueLog Explorer visual verification, scripting	Checks to see if a node's attribute value matches the given value.

Database Value-Retrieving and Verification Functions

Database Value-Retrieving Functions

Database Function	Usage	Description
`RsGetInt`	visual parsing, scripting	Retrieves a number value from the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsGetFloat`	visual parsing, scripting	Retrieves a floating-point value from the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsGetValue`	scripting	Retrieves a binary value from the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsGetString` `RsGetString2`	visual parsing, scripting	Retrieves a string value from the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsIsNull`	scripting	Checks, whether the value from the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it, is null.
`RsRows`	scripting	Retrieves the number of rows in the internal result set,

Database Function	Usage	Description
		which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsCols`	scripting	Retrieves the number of columns in the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.

Database Verification Functions

Database Function	Usage	Description
`RsVerifyRowCount`	visual verification, scripting	Checks whether the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it, has a specific amount of rows.
`RsVerifyInt`	visual verification, scripting	Verifies a number element at a given row/column within the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsVerifyFloat`	visual verification, scripting	Verifies a floating-point element at a given row/column within the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsVerifyValue`	scripting	Verifies a binary element at a given row/column within the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.
`RsVerifyString`	visual verification, scripting	Verifies a string element at a given row/column within the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.

Database Function	Usage	Description
`RsVerifyNull`	visual verification, scripting	Verifies, that an element is null at a given row/ column within the internal result set, which is filled whenever a data-generating SQL statement is executed (SELECT statement), no matter which database API (ORA, ODBC) has generated it.

Oracle Forms Parsing and Verification Functions

Oracle Forms Parsing Functions

Oracle Forms Function	Usage	Description
`OraFormsParseTextValue`	TrueLog Explorer visual parsing, scripting	Returns the value of a text field or text area control.
`OraFormsParseCheckValue`	TrueLog Explorer visual parsing, scripting	Returns the state of a check box.
`OraFormsParseRadioValue`	TrueLog Explorer visual parsing, scripting	Returns the state of an option button.
`OraFormsParseListValue`	TrueLog Explorer visual parsing, scripting	Returns the selected value of a combo box, list box, or pop-up list box.
`OraFormsParseListIndex`	TrueLog Explorer visual parsing, scripting	Returns the index of a selected item in a combo box, pop-up list box, or list box.
`OraFormsParseListEntries`	TrueLog Explorer visual parsing, scripting	Returns the number of elements in a list item, combo box, or pop-up list item.
`OraFormsParseListEntry`	scripting	Returns the value of a combo box, list item, or pop-up list item at a specified index.

Oracle Forms Verification Functions

Oracle Forms Function	Usage	Description
`OraFormsVerifyTextValue`	TrueLog Explorer visual verification, scripting	Verifies a text field or text area.
`OraFormsVerifyCheckValue`	TrueLog Explorer visual verification, scripting	Verifies whether a check box is checked or unchecked.
`OraFormsVerifyRadioValue`	TrueLog Explorer visual verification, scripting	Verifies the value of an option button.
`OraFormsVerifyListValue`	TrueLog Explorer visual verification, scripting	Verifies the selected list value of a combo box, list box, or pop-up list.
`OraFormsVerifyListIndex`	TrueLog Explorer visual verification, scripting	Verifies the selected item in a list control.
`OraFormsVerifyListEntries`	scripting	Verifies the number of list entries in a combo box, list box, or pop-up list.
`OraFormsVerifySelectedTreeItem`	scripting	Verifies the selected tree node of a tree control with a provided value.
`OraFormsVerifyPropString`	scripting	Verifies the properties value.

Oracle Forms Function	Usage	Description
`OraFormsVerifyPropInt`	scripting	Verifies an integer property for a specified control.
`OraFormsVerifyPropByte`	scripting	Verifies a byte property.
`OraFormsVerifyPropPoint`	scripting	Verifies a point property.
`OraFormsVerifyPropRectangle`	scripting	Verifies a rectangle property.
`OraFormsVerifyPropNull`	scripting	Verifies if a property value is NULL.
`OraFormsVerifyPropStringArray`	scripting	Verifies all strings in an array.
`OraFormsVerifyPropByteArray`	scripting	Verifies all elements in a byte array. The Oracle Forms type byte array is mainly used for transferring images from the server to the client.

SAPGUI Parsing and Verification Functions

SAPGUI Parsing Functions

SAPGUI Function	Usage	Description
`SapGuiGetStatusBarText`	TrueLog Explorer visual parsing, scripting	Returns the value of a status bar control.
`SapGuiGetText`	TrueLog Explorer visual parsing, scripting	Returns the value of a control.
`SapGuiGetComboboxEntry`	TrueLog Explorer visual parsing, scripting	Returns the selected value of a combo box control.
`SapGuiIsCheckboxChecked`	TrueLog Explorer visual parsing, scripting	Returns the state of a check box control.
`SapGuiIsRadioButtonSelected`	TrueLog Explorer visual parsing, scripting	Returns the state of an option button control.

SAPGUI Verification Functions

SAPGUI Function	Usage	Description
`SapGuiVerifyCheckbox`	TrueLog Explorer visual verification, scripting	Verifies the value of a check box control.
`SapGuiVerifyTextValue`	TrueLog Explorer visual verification, scripting	Verifies the value of a control.

SAPGUI Function	Usage	Description
`SapGuiVerifyRadioValue`	TrueLog Explorer visual verification, scripting	Verifies the state of an option button control.
`SapGuiVerifyStatusBarText`	TrueLog Explorer visual verification, scripting	Verifies the status bar text.
`SapGuiVerifyComboboxEntry`	TrueLog Explorer visual verification, scripting	Verifies the selected entry of a combo box control.

Citrix Parsing and Verification Functions

Citrix Parsing Function

Citrix Function	Usage	Description
`CitrixParseText`	TrueLog Explorer visual parsing, scripting	Returns the text displayed in the specified screen area.

Citrix Verification Function

Citrix Function	Usage	Description
`CitrixVerifyText`	TrueLog Explorer visual parsing, scripting	Verifies the text displayed in the specified screen area.

Terminal-Emulation Parsing and Verification Functions

Terminal-Emulation Parsing Functions

Terminal Emulation Function	Usage	Description
`WebTelnetScreenGetText`	TrueLog Explorer visual parsing, scripting	Queries rendered screen content in an active Telnet connection.
`WebTelnetScreenGetField`	TrueLog Explorer visual parsing, scripting	Queries a rendered screen’s field value in an active Telnet connection.
`WebTelnetScreenGetStatus`	TrueLog Explorer visual parsing, scripting	Queries rendered screen’s status value in an active Telnet connection.

Terminal-Emulation Verification Functions

Terminal Emulation Function	Usage	Description
`WebTelnetScreenVerifyText`	TrueLog Explorer visual verification, scripting	Verifies rendered screen content in an active Telnet connection.
`WebTelnetScreenVerifyField`	TrueLog Explorer visual verification, scripting	Verifies field content in a rendered screen in an active Telnet connection.
`WebTelnetScreenVerifyStatus`	TrueLog Explorer visual verification, scripting	Verifies the status value of a rendered screen in an active Telnet connection.

Index

A

B

C

D

E

F

G

H

P

R

S

T

U

V

W

X