HP Virtual User Generator

For the Windows operating system

Software Version: Service Pack 11.51

User Guide

Document Release Date: December 2012

Software Release Date: December 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Disclaimer for PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format.

Note: Some topics do not convert properly to PDF, causing format problems. Some elements of online help are completely removed from the PDF version. Those problem topics can be successfully printed from within the online help.

Contents

Welcome to the Virtual User Generator User Guide

Welcome to the HP Virtual User Generator, VuGen, HP's tool for creating Vuser scripts. You use VuGen to develop a Vuser script by recording a user performing typical business processes. The scripts let you emulate real-life situations.

You use the scripts created with VuGen in conjunction with other products— HP LoadRunner, HP Performance Center, and HP Business Availability Center.

HP LoadRunner, a tool for performance testing, stresses your entire application to isolate and identify potential client, network, and server bottlenecks.

HP Performance Center implements the capabilities of LoadRunner on an enterprise level.

HP Business Availability Center helps you optimize the management and availability of business applications and systems in production.

LoadRunner Help

LoadRunner Help is an online help system that describes how to use Virtual User Generator. You can access the LoadRunner Help in the following ways:

- Click **Documentation Library** in the Virtual User Generator **Help** menu to open the LoadRunner Help home page. The home page provides quick links to the main help topics.
- Click F1 on any window or dialog box to open the LoadRunner Help to the topic that describes the currently displayed screen.

Topic Types

The content in the above mentioned LoadRunner guides is organized by topics. Three main topic types are in use: **Concepts**, **Tasks**, and **Reference**. The topic types are differentiated visually using icons.

Topic Type	Description	Usage
Concepts	Background, descriptive, or conceptual information.	Learn general information about what a feature does.

Topic Type	Description	Usage
Tasks	**Instructional Tasks**. Step-by-step guidance to help you work with the application and accomplish your goals. Task steps can be with or without numbering: • **Numbered steps**. Tasks that are performed by following each step in consecutive order. • **Non-numbered steps**. A list of self-contained operations that you can perform in any order.	• Learn about the overall workflow of a task. • Follow the steps listed in a numbered task to complete a task. • Perform independent operations by completing steps in a non-numbered task.
	Use-case Scenario Tasks. Examples of how to perform a task for a specific situation.	Learn how a task could be performed in a realistic scenario.
Reference	**General Reference**. Detailed lists and explanations of reference-oriented material.	Look up a specific piece of reference information relevant to a particular context.
	User Interface Reference. Specialized reference topics that describe a particular user interface in detail. Selecting Help on this page from the Help menu in the product generally open the user interface topics.	Look up specific information about what to enter or how to use one or more specific user interface elements, such as a window, dialog box, or wizard.
Troubleshooting and Limitations	**Troubleshooting and Limitations**. Specialized reference topics that describe commonly encountered problems and their solutions, and list limitations of a feature or product area.	Increase your awareness of important issues before working with a feature, or if you encounter usability problems in the software.

Additional Online Resources

The following additional online resources are available from the Virtual User Generator Help menu:

Resource	Description
Troubleshooting & Knowledge Base	Opens the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. Choose **Help > Troubleshooting & Knowledge Base**. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.

Resource	Description
HP Software Support	Opens the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help > HP Software Support**. The URL for this Web site www.hp.com/go/hpsoftwaresupport. • Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. • To find more information about access levels, go to: http://h20230.www2.hp.com/new_access_levels.jsp • To register for an HP Passport user ID, go to: http://h20229.www2.hp.com/passport-registration.html
HP Software Web site	Opens the HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. Choose **Help > HP Software Web site**. The URL for this Web site is www.hp.com/go/software
Add-ins Page	Opens the Virtual User Generator Add-ins Page, which offers integration and synchronization solutions with HP and third-party tools.

What's New in LoadRunner 11.50

Improved VuGen

The VuGen user interface has been improved and enhanced to provide a more flexible and customizable user experience. An all new IDE makes LoadRunner scripting easier and more productive.

Enhancements include:

- New look and feel – Flexible panes, layouts, and more.
- Solution Explorer – An easier way to group multiple scripts, access script items, and perform script-related operations.
- Snapshots – Multiple views, improved performance, snapshot synchronization, and search functionality.
- Improved editor – Context-sensitive support, code completion support, and provides enhanced coloring and formatting.
- Debugger – Real C language debugger.
- Search and replace – New capabilities include search in log and snapshots.
- Step navigator – The new Step Navigator replaces the tree view, providing a single view of the script with easy filtering and search mechanisms.
- New panes – Errors, Tasks, and Bookmarks panes display all errors, messages, and tags in simple, easy to locate views.

- Join the Community – Community integration provides easy access to conversations and threads in the HP Software Community.

For details, see "VuGen Main User Interface - Overview" on page 41

Ajax TruClient Firefox Enhancements

This ground breaking protocol has been enhanced to support the latest technologies, and includes many usability improvements in response to customer needs.

The Ajax TruClient Firefox protocol has been improved with the following enhancements:

- Moving to Firefox 8 for improved performance.
- HTML5 support.
- Create functions to easily reuse and share code.
- Think-time recording.
- Event handlers to support asynchronous behavior.
- API for URL filtering.
- API for setting HTTP headers.
- Automatic transactions per step.

For details, see "Ajax TruClient Protocol Overview" on page 396

Ajax TruClient Internet Explorer

A new protocol that brings TruClient capabilities to Internet Explorer 9. This new addition to the Ajax TruClient family expands support to Internet Explorer (IE) based applications.

For details, see "Ajax TruClient Protocol Overview" on page 396

Web Protocol Asynchronous Support

Support for asynchronous behavior has been added. This new capability enables recording of Poll, Long Poll, and Push interactions.

Advanced Web applications contain a lot of asynchronous communication designed to keep information current and up to date. Applications such as chat, messaging, stock tickers, and news updates all use various asynchronous mechanisms such Poll, Long Poll, and Push to maintain data. Support for these mechanisms has been added to Web (HTTP/HTML) Vuser scripts, and web-based functions inside Flex, and Web Services Vuser scripts. These unique communication patterns are recognized automatically and the recorded script is updated accordingly.

For details, see "LoadRunner Support for Asynchronous Communication - Overview" on page 175

Improved Correlations

A new response correlation capability has been added so correlation is easier and faster. Correlations can now be found based on server responses during recording, in many cases eliminating the need to replay iteratively to find dynamic values. Coupling this with the new Design Studio interface and new APIs for locating parameters based on XPath and Regular Expressions, makes scripting easier and faster.

For details, see "Correlation Tab [Design Studio] Overview" on page 134

Flex Enhancements

Various enhancements have been added to better support this important environment.

Flex enhancements include:

- Using Web correlation mechanisms (rules, studio, response based correlation, web correlation APIs).
- Supporting Web requests.
- Adobe Flex platform jars bundled in the product so application jars are not needed for message serialization.
- RTMP FMS streaming support.
- RTMPT & RTMPS support.
- GraniteDS support.

For details, see "Flex (RTMP/AMF) Protocol" on page 512

Mobile Protocols

New protocols enabling you to develop scripts for mobile applications. Traffic based analysis is used for native applications and Ajax TruClient technology is used for browser based mobile applications.

For details, see "Mobile Protocols Overview" on page 563

Data Format Extension (DFE) for Google Web Toolkit (GWT)

Web protocol includes a built-in DFE to support decoding and encoding of GWT information exchanged as part of GWT remote procedure calls. This facilitates easy correlation and parameterization of GWT based Web applications. The DFE feature is designed to help ease scripting of applications that exchange formatted data. By turning the formatted data into a more readable format the script can be easily correlated and parameterized. GWT DFE is the latest addition to the already supported formats of Base64, JSON, URLEncoding, XML, and Prefix-Postfix.

GWT support includes:

- Format the GWT RPC data into readable and correlate-able content.
- Expose more data, such as object field names.
- Enable simple parameterization.
- Solve specific correlations for GWT.

For details, see "Google Web Toolkit - Data Format Extension (GWT-DFE) - Overview" on page 675

.NET4 Support

The .NET protocol now supports .NET4 framework.

.NET4 support has been added to the existing support for .NET frameworks 2-3.5.

Web Services Enhancements

The Web Services protocol includes new features for better security support, improved handling of

WCF, as well as additional improvements based on customer feedback.

The Web Services protocol includes the following enhancements:

- Improved interface for security settings, including addressing versions.
- Easier certificate selection.
- Flexible definition of signature and encryption algorithms. Also includes the option to exclude timestamps.
- Support of custom extensions to WCF.
- Support of LoadRunner HTTP capabilities for WCF.

For details, see "Set Security Properties Dialog Box" on page 780 and "WCF Extensibility" on page 773

Integrating Virtualized Services

Integrate with HP Service Virtualization, and use simulated services, to facilitate load testing business processes that contain services that are not readily available or too costly. As part of your performance test, you may want to test applications that depend on other services which are a part of your business scenario. Instead of loading actual services, you can use simulated services in your test run. The virtualized services are a simulation of actual services. To facilitate performance testing business processes that contain services that are not available, Performance Center integrates with HP Service Virtualization. Using simulated services in your test is helpful if using actual services involves an additional cost or requires the service of an application that is under development or inaccessible at the time when you run your performance test.

For details, see HP Service Virtualization Integration - Overview.

Recording 64bit Applications

Recording of 64bit applications has been added to the existing support for 64bit operating systems. 64bit applications can usually be recognized when the '*32' suffix is not displayed in the Windows Task Manager for their process. Replay is in 32bit.

IPv6 support

IPv6 based applications can be tested in addition to IPv4 based ones. Support includes IP Spoofing.

Note: Internal LoadRunner communication, such as Controller Load Generator, is still IPv4 based.

What's New in LoadRunner Service Pack 11.51

LoadRunner Service Pack 11.51 contains many new features and enhancements that significantly improve LoadRunner functionality and your user experience.

The new features and enhancements are described below:

Improved VuGen

The VuGen user interface has been further improved and enhanced to provide a more flexible and customizable user experience.

Enhancements include:

- Snapshots - A new compare snapshot feature that enables you to split the snapshot pane and view and compare the record snapshot with the replay snapshot.
- An improved Summary Report - That now includes detailed information from all iterations.
- Design Studio – Improved correlation functionality including:
  - A discard to correlation feature that enables you to delete the selected dynamic values from the correlation grid when a dynamic value has a status of new.
  - Design Studio supports correlation for the RTMP/T protocol.
- Conditional Breakpoints – Enhanced conditional breakpoints functionality.
- Thumbnail Explorer and Thumbnails - The new Thumbnail and Thumbnail Explorer features enable you to follow the recorded business process in a visual manner. The Thumbnail Explorer feature enables you to navigate to a location in the editor based on a visual representation of a step. You can also navigate in the Editor and view the corresponding visual content of the step in the Thumbnail Explorer.

Ajax TruClient

Ajax TruClient has been significantly enhanced as follows:

- Ajax TruClient for IE Client – The Ajax TruClient for IE now supports a wealth of new features in-line with the support previously provided for the Ajax TruClient for FF client. For example, function libraries and event handlers.
- Ajax TruClient now has an improved and more user friendly user interface.
- Additional JavaScript APIs have been provided.
- TruClient Engine - The engine has enhanced performance, and improved heuristics have been implemented.
- A new replace with parameter feature has been added.
- Ajax TruClient documentation is now more intuitive and feature specific.

Improved Correlations

Correlation capabilities have been improved with the following:

- An argument has been added to the web_reg_save_param_xpath API that instructs the query to return the entire content of the selected node, and not just its value.
- The web_reg_save_param_regexp API now allows you to enter multiple capture groups in the regular expression and specify which capture group to save.You can also now save the entire matched expression.
- New out-of-box correlation rules for ASP.NET.
- You can now change the boundaries/regular expression/Xpath before correlating.
- Correlation studio can now also correlate the RequestUrl argument.

Administrator Right Restrictions

Virtual User Generator (VuGen) and Analysis can now be run without Administrator privileges.

Expanded Data Format Extension Enhancements (DFE) Support

DFE support for the following has been added:

- Binary XML DFE that transforms Microsoft WCF binary XML into XML format.
- Remedy DFE that transforms Remedy request data into XML format.
- XSS DFE that enables you to test sites that use Cross Site Scripting (XSS) defense code.

Enhanced Web HTTP/HTML Protocol Support

Web HTTP/HTML protocol has been enhanced as follows:

- A new web_set_pac API that enables you to change the PAC setting from within the script.
- The web_set_user API has been enhanced. Now if an empty string ("") is passed for host:port, the username and password in the current web_set_user call are applied to all domains unless a different user and password has been set for the domain with another web_set_user call.
- The maximum number of concurrent connections to the server is automatically configured based on the User-agent string.
- JavaScript Engine – New functionality has been added that enables you to run JavaScript code in Web HTTP/HTML scripts.

Flex Enhancements

- New support for AMF polling and long polling. This allows the recognizing and replaying of Polling and Long Polling in flex_amf_call steps in Flex Vuser scripts.
- RTMP/T now supports automatic correlations (Studio, rules, Response based correlations).
- The flex_rtmp_receive_stream and flex_rtmp_tunneled_send APIs now allow you to specify a time after which a script will continue to the next step.

Enriched Citrix Support

Citrix support has been expanded as follows:

- XenApp enhancements:
  - The following APIs have been added - Ctrx_Logoff: Closes the current Citrix session, and Ctrx_Get_Server_Name: Returns the Citrix server name.
  - Support for XenApp 6.5 has been added.
- ICA file adjustment - Enables support engineers to tweak the ICA files received during Citrix ICA + NFuse recording/replay without having to make changes on the Citrix Server Web Interface level.
- Support for Citrix XenDesktop has been added.
- Citrix Access Gateway support: LoadRunner supports CAG for Citrix Client version 10.200 (or less) and Citrix Client version 13.x.

IPv6 Support

The following IPv6 support has been added to the currently supported monitors and utilities:

- NDM (Network Delay Monitor)
- Webtrace

Java over HTTP (JOH) enhancement

Asymmetric java objects traffic support has been added.

Oracle NCA

Support for Oracle forms 11 has been added.

Analysis

Analysis has been enhanced as follows:

- SQLite - New support for SQLite embedded database has been added. SQLite supports a load results database of up to 32 TB.
- Color Palette - A new color palette feature has been added, enabling you to specify up to 100 colors, and to select colors that are associated with graph entities.
- Analysis Graphs - You can now add graph notes to rich reports.

VuGen

Welcome to the HP Virtual User Generator, VuGen, HP's tool for creating Vuser scripts. You use VuGen to develop a Vuser script by recording a user performing typical business processes. The Vuser scripts let you emulate real-life situations.

You use the scripts created with VuGen in conjunction with other products— HP LoadRunner Controller, HP Performance Center, and HP Business Availability Center.

HP LoadRunner, a tool for performance testing, stresses your entire application to isolate and identify potential client, network, and server bottlenecks.

HP Performance Center implements the capabilities of LoadRunner on an enterprise level.

HP Business Availability Center helps you optimize the management and availability of business applications and systems in production.

Working with VuGen

Main VuGen User Interface

VuGen Main User Interface - Overview

The VuGen Main User Interface is comprised of several zones, each of which can display a variety of panes. You can modify the layout of these zones and panes to enhance recording and debugging of your Vuser scripts. For details, see "VuGen Layouts - Overview" on page 48.

The following table describes each pane and provides a short use case scenario.

Pane	Purpose	Use Case Scenario
Bookmarks	Bookmarks allow you to specify a location in a script so that you can easily find it later on for editing. For details, see "Bookmarks Pane" on page 92.	

Pane	Purpose	Use Case Scenario
Errors	Displays script errors, warnings and messages generated from script replay. Additionally, you can create custom filters for error messages. For details, see "Errors Pane" on page 94.	**Error Filter from the output log** After every test process, such as code generation and replay, you can check the error pane for the error log. You can also view other types such as warning and message. Community search is available with context menu on highlighted error. In addition, you can double click the message to jump to the location in the script.
Snapshot	A snapshot displays server and client data associated with a specific step in a script. The format of the data is dependent on the protocol used for creating the script. For Snapshot details, see "Snapshot Pane" on page 90.	Use the snapshot pane to understand all data that the steps contain. You can perform certain tasks such as search for correlations, compare record versus replay snapshots and search for the specific values using the standard search operation.
Data Grid	Simplified view of all the recordsets associated with the script. Valid for specific protocols such as MSSQL.	The data pane contains either sent or received data. The data is displayed in an easy formatted table and you perform operations such as parameterization, and other data manipulations.

Pane	Purpose	Use Case Scenario
Solution Explorer	The Solution Explorer enables you to organize and manage multiple scripts in a named solution. This pane provides easy access and manipulation of solutions such as script assets, parameters, runtime settings, and replay runs. You can double-click an asset to activate it in the editor area or right-click to examine quick operations available for that asset. For details, see "Solution Explorer Pane" on page 84.	You can now bundle scripts in a solution. For example you can bundle scripts related to one business process. For the present, the solution entity is limited in the following ways: • The solution entity is limited to local script development such as save all or open. • The solution entity can not be imported into any of the existing management tools such as ALM or Controller.

Pane	Purpose	Use Case Scenario
Step Navigator	Enables you to navigate to a selected step in your script. If your script contains many steps, you can use the search box to search for matching text in the different parts of the steps. For details, see " Step Navigator Pane" on page 93.	Table view of all LoadRunner API steps that exist. The step navigator replaces the tree view in previous versions of VuGen. • Dynamic filter on various step properties such as step arguments. • You can view the script in either action or script scope. • Every step that has a snapshot is marked with an icon. When hovering over a step that has an associated thumbnail it is presented as a tooltip. • Double clicking a step, takes you to the corresponding location in the script and synchronizes all other panes. • Various operations can be done from the context menu such as copy stop. • Step Navigation is synchronized based on the validity of the script. You can check the status of the pane during script editing.

Pane	Purpose	Use Case Scenario
Tasks	You can add, edit or search for tasks related to a script or solution. For details, see "Task Pane" on page 95.	**Centralized Task Management** During script development, comments tasks enable you to embed tasks in the script. For example in a C based language script, `//TODO Add Parameter` User tasks enable you to add tasks, assign ownership and track completion of tasks related to the overall script process. For example, **Add Load Test Scenario using newly created script**.
Editor	The editor area contains all open assets such as script actions, extra files,report summary enables you to edit script actions. In addition you can open community search, help documentation, and browser pages. For details, see "Editor" on page 78.	You can view script actions simultaneously from different scripts within a solution. • You can insert comment blocks. • You can customize the color scheme. • You can enlarge the fonts. • Auto-completion options using CTRL + SPACE. • All supportive data in the panes is synchronized as you navigate through the script. • The script support full debug capability can be managed within the script such as toggle breakpoints. • Community search is available on highlighted text in the script.

Pane	Purpose	Use Case Scenario
Thumbnail	Enables you to visually follow the business process that the script has recorded. For details, see "Thumbnail Explorer" on page 79.	• The thumbnail generation is configurable in **Tools > Options.** • You can enlarge the thumbnail to full size by double clicking. • Additional operations are available from the context menu such as **Go to Step.**
Output	Event log from different operations in VuGen such as code generation, replay, and recording. For details, "Output Pane" on page 89	The output pane displays all logs created during script development phases such as code generation, replay, and recording. • You can perform quick searches from the pane or access full search with CTRL+F • You can save the logs.
Breakpoints	VuGen lets you include breakpoints in you Vuser scripts to help you to debug the scripts. For details, see "Breakpoints Pane" on page 96.	Enables you to set and manage breakpoints to help analyze the effects of the script on your application at pre-determined points during script execution.
Watch	The Watch pane enables you to monitor variables and expressions while a script runs, and is in the **Paused** state. For details, see "Watch Pane" on page 99.	
Call Stack	This debug pane enables you to view information about the methods and functions that are currently on the call stack of your script, or the context in which the run session was paused. For details, see "Call Stack Pane" on page 100.	

For a quick summary, see "Performing Common Tasks in the New VuGen Interface" on next page.

Performing Common Tasks in the New VuGen Interface

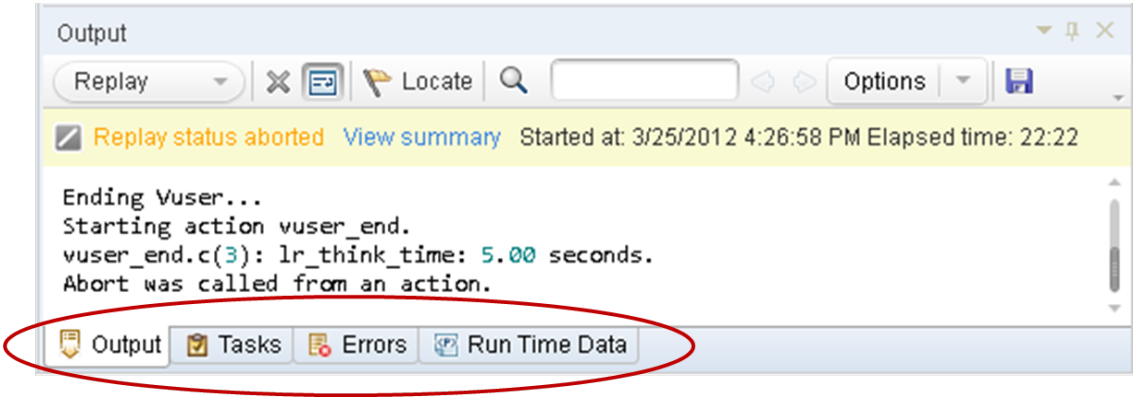
VuGen's user interface has been updated for version 11.50. Some tasks are not performed in the same way as they were in earlier versions and some new tasks have been added. The table below helps you find the new way of performing tasks that have changed. For more details, see the linked topics. For a full description of the new interface, see "VuGen Main User Interface - Overview" on page 41.

Feature/Function	Old VuGen	New VuGen	Links
Solution	Scripts were independent of each other and needed to be combined manually.	Scripts can now be combined in a solution.	"Solution Explorer - Overview" on page 50
Script naming	New scripts were kept in a temporary directory and only had to be named when saving them.	New scripts are created directly in the provided path and should be named when creating them.	
Script location		The default script location has changed. It is now C:\Users\<UserName>\Documents\VuGen\Scripts.	
Access to Run time settings	Run time settings were accessed from the toolbar and menus.	Run time settings are now accessed from the Solution Explorer pane.	"Solution Explorer Pane" on page 84
Access to parameters	Parameters were accessed from the toolbar and menus.	Parameters are now accessed from the Solution Explorer pane.	"Parameter List Dialog Box" on page 254
Tree view	The tree view enabled you to view the structure of the steps in your script.	The tree view has been replaced by the Step Navigator.	" Step Navigator Pane" on page 93
Snapshots		You can now see Snapshots together with the relevant script in a multi-pane window.	"Snapshot Pane" on page 90
Open script directory	Script directories were opened from a context menu within the script.	Script directories are now opened from a context menu on the Script tab.	"Script Directory Files" on page 109

Feature/Function	Old VuGen	New VuGen	Links
Open multiple script files	Only one script file could be opened to be viewed. If you viewed a new script file, the previous script file would close.	Multiple script files can be opened and viewed simultaneously	"Editor" on page 78
Compare scripts	To compare scripts you selected **Tools > Compare with Script** and a select dialog box opened directly to the script directory.	To compare scripts you now select **Tools > Compare > Compare with external object**. A dialog box opens and you navigate to the appropriate directory.	"How to Compare Scripts Side by Side" on page 60
Inserting new steps	To insert new steps in a script you selected **Insert > New Step** and a popup opened to enable you to insert your new step.	To insert a new script you now select **Design > Insert in Script > New Step**. The Steps Toolbox pane opens with steps for you to select.	"How to Insert Steps into a Script" on page 207
Multiple layouts		You can now choose from a number of layouts depending on the task you are performing. You can also customize your layout to suit your needs.	"VuGen Layouts - Overview" below
Debugger		New options have been added to Debug: Watch and Call Stack.	"Watch Pane" on page 99 "Call Stack Pane" on page 100

VuGen Layouts - Overview

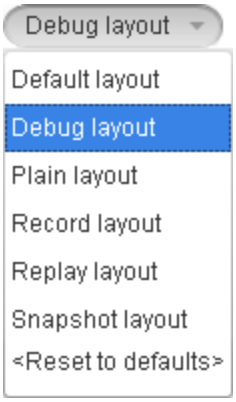
The VuGen window is composed of a number of zones. Each zone can contain a variety of panes, such as the Errors pane or the Snapshots pane. The panes appear as tabs within the zones. The figure below shows a zone that contains four tabbed panes: Output, Tasks, Errors, and Run Time Data. The Output pane is currently visible.



The specific configuration of the zones and the panes contained within the zones is called a layout. VuGen is supplied with a set of standard layouts: **Default**, **Debug**, **Plain**, **Record**, **Replay**, and **Snapshot**. Each layout is designed to enhance a specific phase of the Vuser script development process. For example, the Replay layout includes the panes that are most useful when you run a Vuser script: **Errors**, **Call Stack**, **Watch**, **Breakpoints**, **Output**, and **Run Time Data**. VuGen automatically uses specific layouts during specific phases of the script development process. For example, the Record layout is used while you record a script, and the Replay layout is used when you replay a script.

The VuGen toolbar displays the layout that is currently used 

. To change the layout, click the **Layout** drop down and select the required layout from the list of layouts, as shown below.



You cannot add or delete a standard layout. However, you can modify most of VuGen's standard layouts to meet your specific requirements. When you modify a layout, you can add, move and resize zones, select which panes to include in each zone, and specify which of these panes is displayed by default. For task related details, see "How to Modify the VuGen Layout" on page 62. After you modify a standard layout, VuGen maintains that layout until you change the layout again or reset the default layouts.

Note: VuGen does not save any changes that you make to the Plain layout.

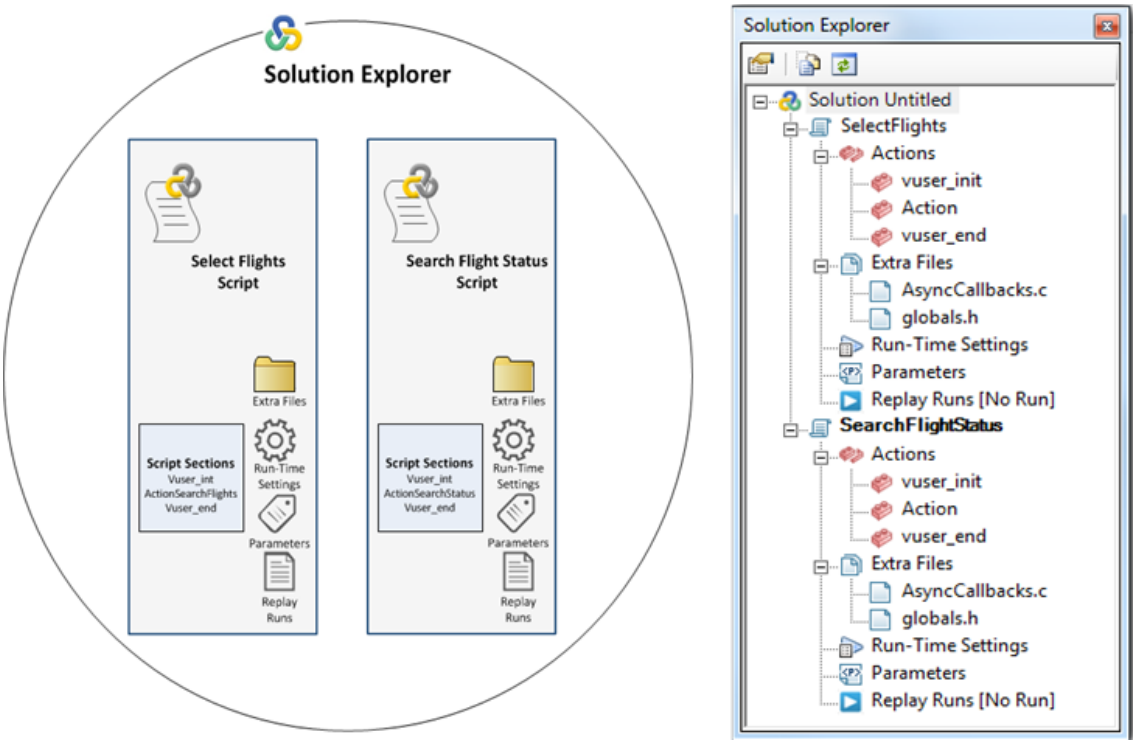
Restoring the layout defaults

On the VuGen toolbar, click the **Layout** drop-down, and select **Reset to Defaults**. VuGen resets all standard layouts to their default settings.

Solution Explorer - Overview

The Solution Explorer enables you to easily organize and navigate through script entities, enhancing the recording, replay and debug process. You can create a solution containing multiple scripts of different protocols related to a full-cycle business process. Each script entity includes extra files (such as header files), run-time settings, parameters, and replay runs.

The following graphic illustrates the Solution Explorer structure:



Solution

When you create a new script, you can name the solution that contains the script. For example, you can use the business process as the solution name. If you do not specify a name, the default name is "Untitled".

Scripts

Your solution can contain multiple scripts. Single-click a script, or one of its assets, to change focus to that script. When a script in a solution is in focus, it changes the behavior of VuGen. For example, when you click replay, the script in focus runs. In addition, menus options, toolbars buttons, and panes display functionality relevant to the script's protocol. For example, the script in focus is recorded in Web HTTP/HTML , the **Recording Options** button is displayed on the toolbar. However, if the script in focus is recorded in Ajax TruClient, the **Develop Script** button is displayed on the toolbar.

Double-click the script's action to open it in the Editor.

In addition, you can drag and drop scripts (`<scriptName>.usr`) from the file directory to the **Solution Explorer**.

Extra files

Additional files that are called by the script, are contained in the Extra Files node under **Solution Explorer > <ScriptName>**. You can drag and drop header files (`<headerFileName.h>`from your file directory. When you include files in the Extra Files node, these files will automatically be included in a Load Generation Scenario.

The information contained in extra files can include:

- Common utility functions used by the script (for example, code)
- Definition of constants and variables used by the script (for example, code)
- Special assets used during script execution (such as .jpeg files)
- Data files manipulated by script code during script execution

The following are examples of file types that can be added as extra files:

.ws,.h,.c.,dat,.ini,.vbs,.java,.js,.txt,.tux,.rec,.msc,.vdf,.xml,.xsl,.dtd,.html,.htm

You can edit extra files in the editor if the file type is included in **Tools > Options > Script Management > List of file types that can be edited in the Editor**. Double click the extra file to open it in the Editor. For details on how to modify the list, see "Scripting Options" on page 71.

Run-time Settings

You can access run-time settings for a specific script from the Run-time Settings node in the **Solution Explorer > <Script> > Run-Time Settings**. For details, see "Run-Time Settings" on page 332.

Parameters

You can access parameters for a specific script from the Parameters node in the **Solution Explorer > <Script> > Parameters**. For details, see "Parameter Overview" on page 228.

Replay Runs

This node enables you to access the **Replay Summary Reports** for each iteration in the replay. For details, see "Solution Explorer Pane" on page 84.

Editor - Overview

VuGen's Editor enables you to edit recorded scripts and other supplementary files such as header files. You can open multiple files simultaneously, easily navigating tab by tab.

The Editor facilitates editing by providing the following features:

Tab View

The tab view in the Editor allows you to organize and edit several script assets simultaneously. You can do the following in the Editor:

- Double-clicking on a script or extra files in the Solution Explorer will open the file in Editor area as a tab. The tab name is **`<script name>:<filename>`**
- Drag tabs to reorder them.

Support of Multiple Programming Languages

Recorded scripts are generated in C, which has full language and parsing support in VuGen. You can enhance your Vuser scripts by adding standard ANSI C functions. ANSI C functions allow you to add comments, control flow statements, conditional statements to your Vuser scripts. You can add standard ANSI C functions to any type of Vuser script. For details, see "C Vuser Scripts" on page 806.

In addition, the VuGen Editor enables you to write manual scripts with the following programing languages:

- VB Script

  For details, see "VBScript Vusers" on page 807.

- Java

  For details, see "Java Vusers" on page 807.

- JavaScript

  For details, see "JavaScript Vusers" on page 805.

- C#

  For details, see ".NET Vusers" on page 808.

- VB.NET

  For details, see "How to Create a Vuser Script with Visual Basic or Visual C#" on page 811.

Code Completion and Tooltips for C Scripts

Code completion enables you to quickly and accurately write code by providing a list of code items from which you can select . Press **CTRL + SPACE** to activate statement completion when your cursor is in the Editor. Tooltips, containing context information, appear when your mouse hovers over a code element. The following table describes available code completion items, scope, identifying icon, and tooltip context:

Code Completion Item	Scope	Icon	The Tooltip Displays
ANSI C Keywords and Types	All possible standard C keywords.		Function type, name, and parameters.
LR API Functions	All steps in the script.		LR API Step.
LR API Constants	Used to delimit groups of parameters in steps. For example, ENDITEM		LR API Constant.

Code Completion Item	Scope	Icon	The Tooltip Displays
User Functions	All the functions that you have defined in action files.		• Function type, name and parameters. • When Using Functions (Method Insight) The required arguments, highlighting each argument as you define it moving to the next argument when you enter the delimiter.
Variables	Local variables – visible only in the function where they have been defined. Global variables – defined outside of any function body. Available everywhere in the script.		Type and name.
Parameters	Available only in the function body where they have been defined.		• Parameter type and name • When Using parameters (Method Insight) The required arguments, highlighting each argument as you define it moving to the next argument when you enter the delimiter.

By default, VuGen uses code completion globally. To disable code completion, select **Tools > Text Editor > Code Completion**. Clear the **Enable code completion** features check box.

Script Code-Coloring for C Scripts

To facilitate script writing and debugging, code item types are colored by an identifying background and foreground. The colored text enables you to easily read scripts and scan for syntax errors. Below is a table that provides examples of code item types and their assigned colors.

Code Type	Color Example
Comments	`/* comment */ // comment`
Keywords	`if (a) { } else { }`
Method Parameter Name	`foo( "parameter=value" )`
LoadRunner API	`web_url`
Method Call	`foo()`
String	`char * text = "Hello, World!"`

In addition, you can customize code item types to suit your needs by selecting **Tools > Options > Text Editor > Highlighting**.

Script Folding for C Scripts

Script folding enables you to selectively hide and display sections of a script, making it easier to manage large scripts by viewing only those sections that you are currently editing. For details, see " Editor Options" on page 66.

Community Search

You can perform Web searches from the VuGen toolbar, which opens a browser tab in the Editor. The default Web site is the LoadRunner Forum which enables you to search topics, post questions, or blog about your expertise. You can add additional search sites by selecting **Tools > Options > General > Community**. For details on adding additional sites, see "General Options" on page 64.

Snapshot Pane - Overview

Snapshots contain information about the status of various steps in a Vuser script. VuGen displays snapshots in the Snapshot pane. The snapshots are displayed in various formats, depending on the protocol of the Vuser script. For example, snapshots for RDP Vusers are displayed as graphic images of the remote screen, whereas snapshots for Winsock Vusers are displayed as textual representations of the data buffers that are transferred while the Vuser script is recorded and replayed.

> Note that the Snapshot pane is not available for all Vuser protocols - only specific Vuser protocols give you access to the Snapshot pane.

Snapshots are captured when a script is recorded and when the script is replayed. The Snapshot pane enables you to display the record and replay snapshots. By default, the Snapshot pane shows just a single snapshot. To enable you to compare snapshots, you can split the Snapshot pane to show two snapshots simultaneously. You can split the Snapshot pane either vertically or horizontally. Each section of the split Snapshot pane can show either a record snapshot or a replay snapshot. Typically, you would show a record snapshot together with its corresponding replay snapshot. This enables you to compare the record snapshot with the replay snapshot.

- For details on working with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

Basic Snapshot pane functionality

The Snapshot pane that is displayed for all Vuser protocols includes the same basic functionality. This basic functionality includes the ability to:

- Show one snapshot, or split the Snapshot pane to show two snapshots. You can split the Snapshot pane either horizontally or vertically.
- Show record and replay snapshots.

For details on how to use the basic Snapshot pane functionality, see "How to Work with Snapshots" on page 57.

Synchronizing snapshots

The Snapshot pane that is displayed for some Vuser protocols improves your ability to compare snapshots by "synchronizing" the two snapshots that are displayed in the Snapshot pane. For example, when synchronizing graphic snapshots, if you place the mouse cursor over a specific location in one of the snapshots, VuGen displays a marker at the corresponding location in the other snapshot. In addition, as you move one snapshot vertically or horizontally within the Snapshot pane, VuGen moves the other snapshot accordingly, ensuring that the same section of each snapshot is displayed.

Note that snapshot synchronization is available for only specific Vuser protocols, and for only specific views within these protocols.

Copying snapshots to the clipboard

You can copy an image-based snapshot to the clipboard. This enables you to import the image into a graphics application, where you can analyze and modify the graphic.

For details on how to copy a snapshot to the clipboard, see "How to Work with Snapshots" on page 57.

Note that the "copy snapshot to the clipboard" functionality is available for only RDP, Citrix, and SAP protocols.

Copying snapshot text to the clipboard

You can copy text from a snapshot to the clipboard. You can then paste the text from the clipboard into another application.

For details on how to copy snapshot text to the clipboard, see "How to Work with Snapshots" on page 57.

Note that the "copy snapshot text to the clipboard" functionality is available for only Ajax (Click & Script) and Web (Click & Script) protocols.

Customized Snapshot pane functionality

In addition to the basic Snapshot pane functionality, the Snapshot panes for some Vuser protocols include customized functionality. For example, the Snapshot pane for RDP Vuser scripts lets you display snapshots in either **Full** or **Image** modes; the Snapshot pane for Winsock Vuser scripts lets you display snapshots in either **Text** or **Hex** modes. The controls for the customized functionality can be found in the Snapshot pane toolbars.

Snapshot on error

In addition to showing record and replay snapshots, the Snapshot pane can display snapshots of errors that occurred during the replay of a script. The "snapshot on error" functionality is available for only specific Vuser protocols.

You can generate and display snapshots of errors only if the "snapshot on error" functionality is activated. For details on how to activate the snapshot-on-error functionality, see "How to Work with Snapshots" on page 57.

Comparing snapshots

The **Compare** button in the Snapshot pane enables you to compare two snapshots. To enable the **Compare** functionality, you must first split the Snapshot pane to show two snapshots. By default, VuGen uses the *WDiff* utility to compare snapshots. You can specify an alternative comparison

tool as described in "Scripting Options" on page 71.

Note that the snapshot comparison functionality is available for only the Web HTTP/HTML and Web Services protocols.

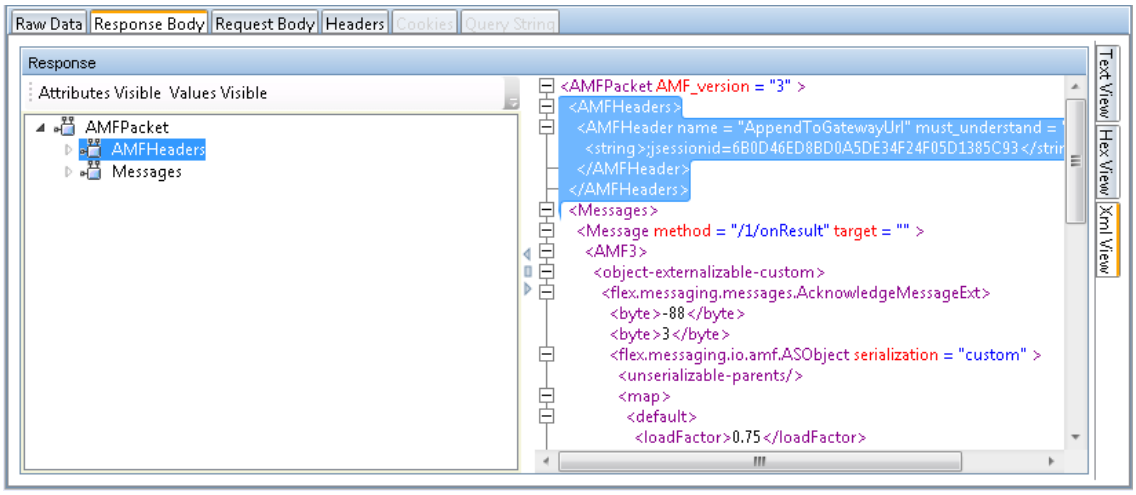
Setting snapshot options

VuGen allows you to set various options that define how snapshots are displayed in the Snapshot pane. For details on these snapshot options, see "Scripting Options" on page 71.

Snapshots that Have an XML View

VuGen's Snapshot pane shows various snapshots that were recorded while a Vuser script was recorded or replayed. For specific Vuser protocols only, the Snapshot pane can show XML views of the snapshot. The XML views are displayed in the XML View tab. The XML View tab appears inside the Response Body and Request Body tabs only. The XML view includes its own set of controls and functionality.

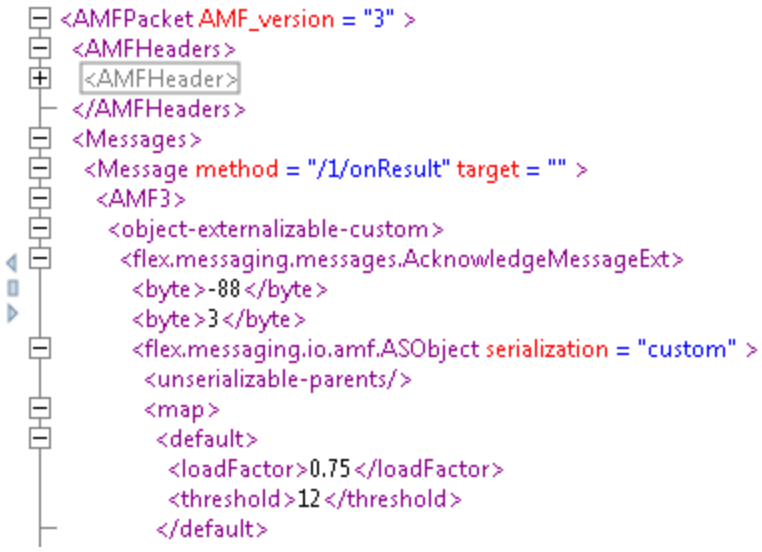
The XML view is divided into two areas. On the left is the tree view of the snapshot data, and on the right is the text view.



The two XML views are synchronized. If you select an entry in the tree view, VuGen highlights the corresponding element, attribute, or value in the text view. Alternatively, if you double-click an element, attribute, or value in the text view, VuGen opens the tree view as required, and highlights the corresponding entry.

The splitter controls ( ) between the tree view and text view enable you to set the proportion of the available space that is occupied by each of the views.

Controls to the left of the text view enable you to expand and collapse elements within the text view. Click an **Expand** icon ⊞ to expand an element in the view, and click a **Collapse** ⊟ icon to collapse an element in the view. Note that you can place the mouse cursor inside an element in the text view and then press the <+> key to expand the element or the <-> key to collapse the element, as appropriate.



Toggle controls [Attributes Visible | Values Visible] above the tree view enable you to define whether or not attributes and values will be visible in the tree view. If both attributes and values are hidden, the tree view shows only the XML elements.

After recording a Vuser script, you can use the XML View in the Snapshot pane to add a text check to the script. For details, see "How to Add a Text Check From the XML View in the Snapshot Pane" on page 59.

How to Work with Snapshots

This topic describes how to use the basic Snapshot pane functionality. For an overview of the snapshot functionality, see "Snapshot Pane - Overview" on page 54.

How to show the Snapshot pane

To show the Snapshot pane, do one of the following options:

- Select **View > Snapshot**.
- Click the **Snapshot** button  on the VuGen toolbar.
- In the Editor, click inside a step that contains a reference to a snapshot.
- In the Step Navigator, double-click a step that contains a reference to a snapshot. Note that in the Step Navigator, each step that contains a snapshot displays a Snapshot icon  . You can

place your mouse cursor over the snapshot icon to see a thumbnail view of the snapshot.

How to show a record snapshot in the Snapshot pane

Click the **Recording Snapshot** button  on the Snapshot pane toolbar.

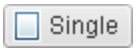
How to show a replay snapshot in the Snapshot pane

Click the **Replay Snapshot** button  on the Snapshot pane toolbar. If more than one replay snapshot exists for the step, click **Iteration** and select the required iteration number.

How to split the Snapshot pane to show two snapshots

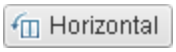
If the Snapshot pane displays only one snapshot, click the **Split Snapshot Pane** button  on the Snapshot pane toolbar.

How to show only one snapshot in the Snapshot pane

If the Snapshot pane displays two snapshots, click the **Single** button  on the Snapshot pane toolbar.

How to toggle between a horizontal split and a vertical split of the Snapshot pane

If the Snapshot pane displays two snapshots:

- Click the **Horizontal Split** button  on the Snapshot pane toolbar to show two snapshots side-by-side.

- Click the **Vertical Split** button  on the Snapshot pane toolbar to show two snapshots, one above the other.

How to synchronize the display of two snapshots when the Snapshot pane is split

1. Make sure that the Snapshot pane is split to show two snapshots.
2. On the Snapshot pane toolbar, click the Synchronization button  .

Note that snapshot synchronization is available for only specific Vuser protocols, and for only specific views within these protocols.

How to copy a snapshot to the clipboard

1. Display the snapshot in the Snapshot pane.
2. Right-click on the snapshot, and then select **Copy Image to the Clipboard**.

Note that the "copy snapshot to the clipboard" functionality is available for only RDP, Citrix, and SAP Vuser scripts.

How to copy snapshot text to the clipboard

1. Display the snapshot in the Snapshot pane.
2. Select the text that you want to copy.
3. Right-click in the selected text, and select **Copy Selection**.

How to activate the snapshot-on-error functionality

1. Click **Replay > Run-Time Settings**. The Run-Time Settings dialog box opens.
2. Under **General**, click **Miscellaneous**.
3. Under **Error Handling**, select the **Generate snapshot on error** checkbox.

How to set the snapshot options

1. Click **Tools > Options**. The Options dialog box opens.
2. Click **Scripting**, and then click **Snapshot**. The snapshot options appear on the right of the dialog box.

How to Add a Text Check From the XML View in the Snapshot Pane

After recording a Vuser script, you can add a text check from the XML view in the Snapshot pane. For details on the XML view in the Snapshot pane, see "Snapshots that Have an XML View" on page 56.

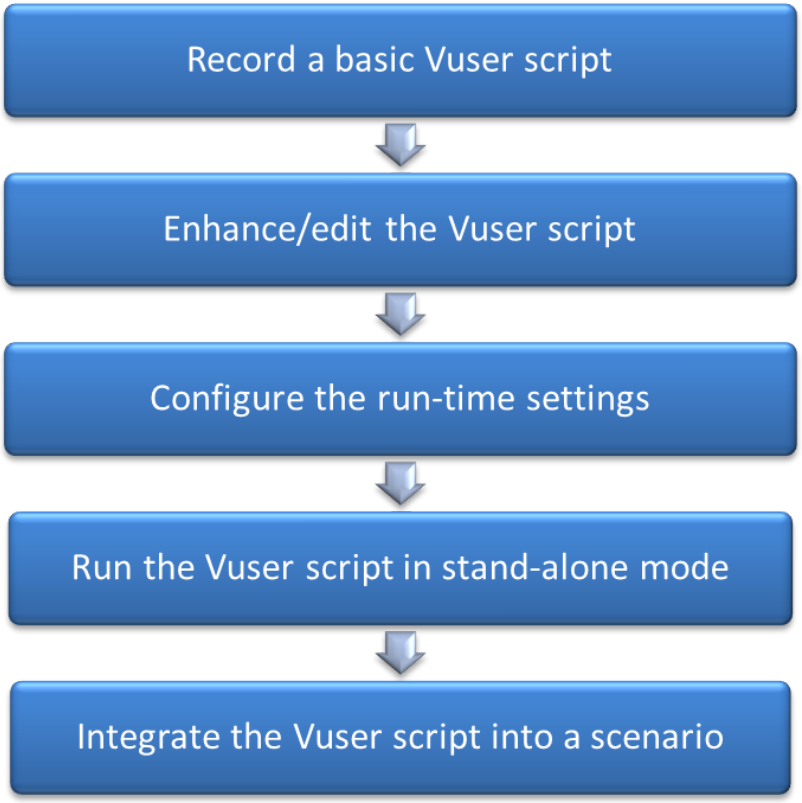
To add a text check from the XML view in the Snapshot pane:

1. Click **View > Snapshot**, or click the **Show Snapshot Pane** button  on the VuGen toolbar.
2. In the Snapshot pane, display a snapshot that contains the text that you want to verify.
3. On the right-side of the Snapshot pane, click the **XML View** tab.
4. In the Snapshot pane, click the **Response Body** tab.
5. In either the Tree view or the Grid view, locate and select the text string that you want to verify.

6. Right-click inside the selection, and select **Add Text Check Step**. The Find Text dialog box opens.
7. Modify the options in the Find Text dialog box. For details on the dialog box options, press F1 when in the dialog box to open the Function Reference.
8. Click **OK** to insert a **web_reg_find** step into the Vuser script.

How to Create a Vuser Script - Workflow

The following diagram outlines the process of developing a Vuser script:



The process of creating a Vuser script is as follows:

1. Record a basic script using VuGen.
2. Enhance the basic script by adding control-flow statements and other LoadRunner API functions into the script.
3. Configure the run-time settings. These settings include iteration, log, and timing information, and define the Vuser behavior during a script run.
4. Verify the script's functionality, by running it in standalone mode.
5. After you verify that the script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile.

How to Compare Scripts Side by Side

Vuser scripts can be compared and displayed side by side using the comparison tool.

To compare Vuser scripts

1. Right click on the primary script in **Solution Explorer** and select **Set as first comparing object**.
2. Right click on the secondary script in the **Solution Explorer** and select **Compare** which will run the compare functionality.

   or

3. Right click and select **Compare with external object**.

   You can compare an asset to a file outside of the solution. This option sets the highlighted asset as the primary asset and opens Windows Explorer to enable you to select the secondary asset.

Note: You can change the comparison tool from **Options > Scripting> Comparison**. For more information, see "Scripting Options" on page 71.

How to Create a Business Process Report

At the final stage of script creation, you can create a report that describes your business process. VuGen exports the script information to one of the following formats:

- Microsoft Word
- Acrobat PDF
- HTML

You can use a pre-designed template or one provided with VuGen, to create reports with summary information about your test run. The VuGen template is available in Microsoft Word 2007 (docx) format. You can edit or update the template according to your requirements.

VuGen lets you customize the contents of the report by indicating what type of information you want to include.

Note: Business Process Reports are available for the following protocols only: Ajax (Click & Script), Ajax TruClient, Citrix ICA, Oracle NCA, Oracle Web Applications 11i, PeopleSoft Enterprise, RDP, SAP (Click & Script), SAP GUI, SAP - Web, Web (Click & Script), Web (HTTP/HTML), and Web Services.

1. ## Create a business process report

   Select **Tools > Create Business Process Report** and complete the dialog box. For user interface details, see "Business Process Report Dialog Box" on page 80.

2. ## Configure additional options

   To modify additional report options such as the table of contents, snapshots, and the document template, click the **More** button. For user interface details, see "Business Process Report Dialog Box" on page 80.

How to Modify the VuGen Layout

The VuGen window is composed of a number of zones. Each zone can contain a variety of panes, such as the Errors pane and the Snapshot pane. When more than one pane is included in a zone, the panes appear as tabs within the zone. This section describes how to customize and modify the zones and panes that appear in the VuGen window.

Moving a pane to a new zone

You can move any VuGen pane to a new zone. The new zone can be either a portion of an existing zone, or it can occupy the entire left, right, top, or bottom of the VuGen window.

In the VuGen window, drag the title bar or tab of the pane that you want to move. (If the required pane is not displayed in the VuGen window, you can select it from the **View** menu.) As you drag the pane over the zones in the VuGen window, a complex marker is displayed in the center of the active zone and a simple marker appears on each edge of the VuGen window.

Note that if you drag the title bar of a zone that contains multiple tabbed panes, then all the panes in the zone are moved to the new zone.

Marker Type	Marker	Description
Complex marker - Current zone		Positions the selected pane in a new zone. The new zone is created in the top, bottom, left, or right of the **active zone**, according to the arrow marker selected when you release the mouse button.
Simple marker - VuGen window		Positions the selected pane in a new zone. The new zone is created in the top, bottom, left, or right of the **VuGen window**, according to the arrow marker selected when you release the mouse button.

Moving a pane to an existing zone

You can move any VuGen pane from one zone to another. When more than one pane is included in a zone, the panes appear as tabs within the zone.

1. In the VuGen window, drag the title bar or tab of the pane you want to move. (If the required pane is not displayed in the VuGen window, you can select it from the View menu). As you drag the pane over the zones in the VuGen window, a complex marker is displayed in the center of the active zone.

2. Locate the cursor over the center button of the complex marker. When you release the mouse button, the selected pane is added as a tabbed pane to the selected zone.
3. Repeat this procedure for each pane you want to move.

Note that if you drag the title bar of a zone that contains multiple tabbed panes, then all the panes in the zone are moved to the selected zone.

Floating and docking panes

Docked panes are fixed in a set position within the VuGen window. For example, when you move a pane to a position indicated by a marker, the pane is docked in that position.

Floating panes are displayed on top of all other windows. Floating panes can be dragged to any position on your screen, even outside the VuGen window. Floating panes have their own title bars.

- To float a pane, right-click the title bar, and click **Float**. The pane opens on top of all the other windows and panes, with its own title bar.
- To dock a pane, double-click the title bar, or right-click the title bar and select **Dock as tabbed document**. The pane returns to its previous position in the VuGen window.

Options Overview

The **Options** dialog box is comprised of VuGen application settings. The settings are common to all protocols available in VuGen.

Specific details of each of the settings are described in the topic of the relevant category topic.

The following table includes a list of the categories and related sections from the **Options** dialog box.

General	• **Task List** enables you to add, update and delete comment tags. • **Projects and Solutions** enables you to configure project settings. • **File Options** enables you to configure save and delete file settings. • **Community** enables you to add, update and delete Community Search Sites. For details, see "General Options" on next page.

Scripting	• **Recording UI** enables you to configure the settings related to the recording user interface, for example the floating toolbar used when recording. • **Replay** enables you to configure the settings related to replay of the script. • **Script Management** enables you to edit the list of file extensions. • **Comparison** enables you to enter the location of the comparing application. • **Step Navigator** enables you to select the background and foreground colors for the **Step Navigator** pane. • **Thumbnails** enables you to configure the **Thumbnail Explorer**. • **Output Pane** enables you to configure the appearance of the **Output Pane**. • **Snapshot** enables you to configure appearance of snapshots in your script. • **Parameters** enables you to define the delimiter to use for parameter braces throughout a script. • **Parser** enables C language parser. For details, see "Scripting Options" on page 71.
Editor	• **General** enables you to configure font and other general options. • **Markers and Rulers** enables you to configure markers and rulers for text used in a script. • **Behavior** enables you to configure settings related to tabs, keystrokes and mouse behavior. • **Code Color** enables you to configure the font settings for each heading, control and text type for each language type. • **Code Completion** enables you to configure the settings for code completion. For details, see " Editor Options" on page 66.

General Options

This pane enables you to configure user interface options.

Task List

To access	**VuGen > Tools > Options > General > Task List**
UI Element	**Description**

, continued

Comment Tags	This pane enables you to add, delete or modify tags names that you can use to label comment tasks in your scripts. **Tags.** List of available tags. **Name.** Displays the name of the highlighted tags in the token list. This area enables you to modify, add or delete the current tag. **Add.** Enables you to add a tag to the token list. **Edit.** Enables you to modify the name of tag from the token list. **Delete.** Enables you to delete a tag from the token list.

Scripts and Solutions

To access	**VuGen > Tools > Options > General > Scripts and Solutions**
UI Element	**Description**
Settings	**Default project location** Enables you to specify a path to your saved projects. Default location = `C:\Users\<username>\Documents\SharpDevelop Projects` **Load previous solution on startup** Enables you to automatically load the previous solution. By default this option is enabled.
Start Page Settings	**Display Start Page on startup** By default this option is enabled. **Close Start Page after script loads** By default this option is enabled.

File Options

To access	**VuGen > Tools > Options > General > File Options**
UI Element	**Description**
Save	**Use temporary file for saving** Temporarily saves files with a .bak extension until save operation is successful. By default this option is enabled.
Delete	**Use recycle bin when deleting files** By default this option is disabled.

Community

To access	**VuGen > Tools > Options > General > Community**
See also	For details, see Community Search in the "Editor - Overview" on page 51.

UI Element	Description
	Community Search Site
	Add a new search site to the list of Community search sites. • **Name**: Enables you to specify a name of the search site that is displayed on the VuGen toolbar. • **URL**: Enables you to specify the URL of the search site. Example URLs http://www.bing.com/search?q=%QUERY% http://www.google.com/search?q=%QUERY% http://www.google.de/search?q=%QUERY% (localized google site, e.g. de for Germany) http://en.wikipedia.org/wiki/%QUERY%
	Enables you to edit the properties of the custom search site.
	Delete the search site from the list of available sites.
	Move the search site lower down on the list of available sites.
	Move the search site higher up on the list of available sites.

Editor Options

This pane enables you to configure the text editor options.

General

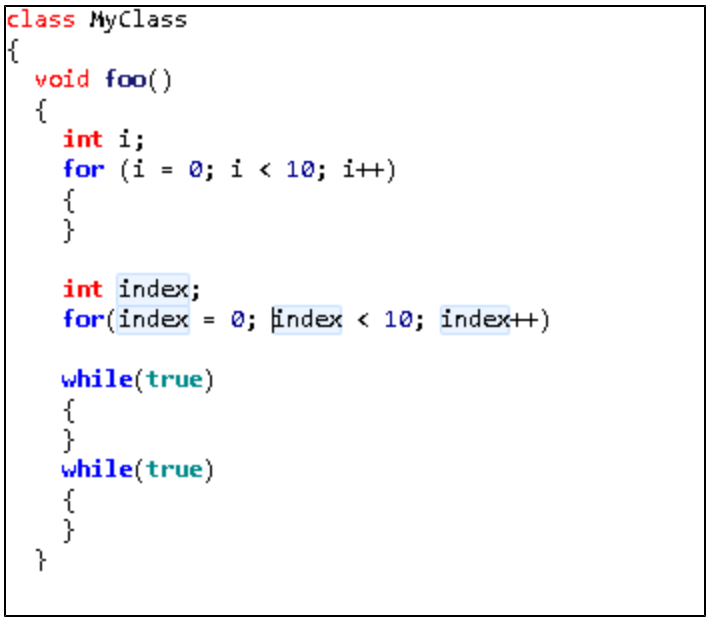
To access	**VuGen > Tools > Options > Editor > General**

UI Element	Description
	Font
Text Font	Enables you to select the font.

Size	Enables you to select the font size.
General Options	
Word wrap	Automatically wraps text to the next line. By default this option is disabled.
Show Class/Function Browser	Show or hide the Class/Function Browser. When enabled, you can navigate quickly to a specific class or function in your script by selecting it from the drop down list in the browser. By default this option is enabled.
Show Line numbers	Enable line numbering of script in the Editor. By default this option is enabled.
Enable working URL hypertext links in the Editor	Enable URLs in scripts to function as hypertext links. Disabling this option may increase performance. By default this option is enabled.

Markers and Rulers

To access	**VuGen > Tools > Options > Editor > Markers and Rulers**
UI Element	**Description**
Markers and Rulers	
Show spaces	Enable markers that indicate where tabs exists. By default this option is disabled.
Show tabs	Enable markers that indicate where line ends. By default this option is disabled.
Show end-of-line makers	Enable markers that indicate where line ends. By default this option is disabled.
Underline errors	Enable underlining of errors. By default this option is enabled.
Highlight matching brackets	Enable highlighting of matching brackets. By default this option is enabled.

Highlight symbols	When this option is enabled, you can select a non-keyword in your script and the Editor will highlight all other occurrences in your script.  By default this option is enabled.

```
class MyClass
{
  void foo()
  {
    int i;
    for (i = 0; i < 10; i++)
    {
    }

    int index;
    for(index = 0; index < 10; index++)

    while(true)
    {
    }
    while(true)
    {
    }
  }
```

Behavior

To access	**VuGen > Tools > Options > Editor > Behavior**
UI Element	**Description**
Tabs	
Indentation	Enables you to set the spacing for tab indentation. Default indentation is four spaces.
Convert tabs to spaces	Converts tabs to spaces. By default this option is disabled.
Use smart indentation	Automatically applies the indentation format from the previous line. By default this option is enabled.
Behavior	
Enable zoom with mouse wheel	Enables you to use the mouse wheel to zoom. By default this option is enabled.

, continued

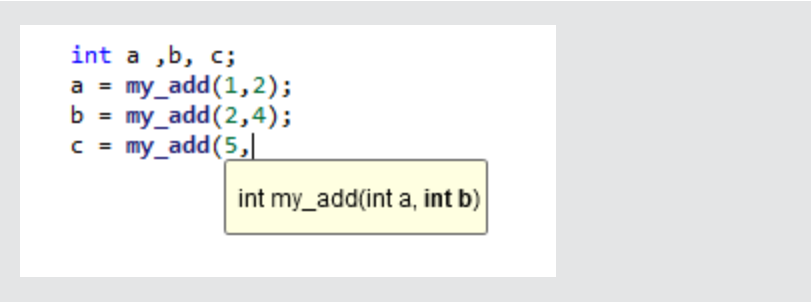
Cut or Copy entire line when nothing is selected	When this option is enabled and the cursor is located in a line of code, you can cut or copy the entire line without highlighting. By default this option is enabled.
Use Ctrl + Click for "Go to Definition"	Enable shortcut for "Go To Definition". This shortcut enables you to move the cursor to the definition of a function you have highlighted in the Editor. By default this option is enabled.

Code Color

To access	**VuGen > Tools > Options > Editor > Highlighting**
Important information	Highlighting options enable you to customize the color of script elements.
UI Element	**Description**
<Language Selection>	Drop-down list of script languages for which you can customize appearance including: • C# • HTML • VuGen C • XML
<Element Selection>	List of code elements whose appearance you can customize.
Foreground color	Enables you to select a color from the pallet. The select color is applied to the foreground of the code element.
Background color	Enables you to select a color from the pallet. The select color is applied to background of the code element.

Code Completion

To access	**VuGen > Tools > Options > Editor > Code Completion**
UI Element	**Description**
Enable code completion features	Code completion features are enabled. For details on code completion, see "Editor - Overview" on page 51. By default this option is enabled.

Enable syntax tooltip	Enables tooltips that display function arguments. Each argument is highlighted as you are defining it, moving to the subsequent argument when the delimiter is entered.  By default this option is enabled.
Show tooltips when mouse pointer stops over an identifier	When this option is enabled, a description of the code elements is displayed when the mouse hovers over the identifier. This option is disabled when **Show tooltips in debug mode only** is enabled. By default this option is enabled.
Show tooltips in debug mode only	Tooltips are displayed in debug mode only. By default this option is disabled.
Include in the code completion list	
ANSI C keywords	Enables you to include ANSI C keywords in code completion list. By default this option is enabled.
LoadRunner API Steps	Enables you to include LoadRunner API Steps in the code completion list. By default this option is enabled.
LoadRunner API Constants	Enables you to include LoadRunner API Constants in the code completion list. By default this option is disabled.
User-defined functions	Enables you to include user-defined functions in the code completion list. By default this option is enabled.
Function parameters, local and global variables	Enables you to include function parameters, local, and global variables in the code completion list. By default this option is enabled.

Folding

To access	VuGen > Tools > Options > Editor> Folding
UI Element	**Description**
Enable Folding Features	This option enables expanding and collapsing of script sections. By default this option is enabled.
Enable Steps folding	This option enables expanding and collapsing of script steps. By default this option is disabled.
When step length is more than [ ] characters	Enables you define the number of characters in a step before implementing folding. By default this option is disabled.
When step consists of more than [ ] lines	Enables you to define the number of lines to a step before implementing folding. By default this option is disabled.

Scripting Options

This pane enables you to configure options related to recording, replaying and debugging scripts.

Recording UI

To access	VuGen > Tools > Options > Scripting > Recording UI
UI Element	**Description**
Enable Recording Floating Toolbar transparency mode	Display a transparent floating recording toolbar. When you click or hover on the toolbar it becomes opaque. By default this option is disabled.
Show Start Recording Dialog Box after new script is created	Automatically open the Start Recording Dialog Box after a new script is created. By default this option is disabled.
Automatically close transactions	Enable this function if you want VuGen to insert an end transaction step for open transactions before recording a subsequent action. By default this option is disabled.

Replay

To access	**VuGen > Tools > Options > Scripting > Replay**
UI Element	**Description**
Animated Run	Animated Run You can run a Vuser script in animated mode or non-animated mode. When you run in animated mode, VuGen highlights the line that is running in the Vuser script. When you run in non-animated mode, VuGen executes the Vuser script, but does not indicate the line being executed. By default this option is enabled. Animated Run Delay You can set a delay of the highlighting in the animated run, allowing you to better view the effects of each step. You set the delay in milliseconds. By default this option is disabled. Animate Functions in Action sections only Animates the content of the Action sections only, not the init or end sections. By default this option is enabled.
Results	Enable result of replay summary to be saved to a named folder after each script run When this option is enabled the dialog box prompts you to name a results file before running a script in VuGen. When not enabled, VuGen automatically names the directory 'result1'. Subsequent script runs will automatically overwrite previous results files unless you specify a different name. **Note:** Results are stored in a subdirectory of the script. By default this option is disabled. **Generate report during script execution** By default this option is disabled.
During Replay	**Show run-time view during replay** Enables the run-time viewer **Auto Arrange Window** Select this option to arrange the two viewers side by side. This option is disabled by default.

, continued

After replay	**After replay show** Instructs VuGen how to proceed after the replay: • **Script.** Show script in the Editor. • **Replay summary.** Go directly to the Replay Summary window in the Editor. (Default)

Script Management

To access	**VuGen > Tools > Options > Scripting > Script Management**
UI Element	**Description**
List of file types, by extension, that can be edited in the Editor	Enables you to modify the list of valid file extensions that can be edited in the Editor.

Comparison

To access	**VuGen > Tools > Options > Scripting > Comparison**
UI Element	**Description**
Path to comparing tool	You can select a comparison tool to be used when comparing two scripts. VuGen is installed with a default comparison tool (Wdiff). Click the browse button to locate a third-party comparison tool that is installed on your local machine.
Command line arguments	This option contains the mandatory default arguments %1 and %2 for the two files you want to compare, which should not be modified. You can add additional arguments, as needed, for your comparison tool.

Step Navigator

To access	**VuGen > Tools > Options > Scripting > Step Navigator**
UI Element	**Description**
Enable Editor highlighting	This option enables you to highlight filtered steps in your script.
Background color	Enables you to select a background color to apply to the filtered steps in your script.
Border color	Enables you to select a border color to apply to the filtered steps in your script.

Thumbnails

To access	VuGen > Tools > Options > Scripting > Thumbnails
UI Element	**Description**
Enable Thumbnail Explorer	Enables slide view of generated thumbnails in the Thumbnail Explorer. Double clicking thumbnail sets the cursor at the associated step in the script. By default, this option is disabled.
Highlight the thumbnail associated with a step	Sync the display of the Thumbnail Explorer while scrolling through steps in the Editor.
Show important thumbnails by default	VuGen displays thumbnails directly related to the business process and filters out less important thumbnails by default.
Enable automatic creation	Enables the automatic creation of thumbnails during the application's idle time.
Cache thumbnails to script folder	Optimize VuGen's performance by saving rendered thumbnails to a cache file. Thumbnails are loaded from the cache file after the initial generation.

Output Pane

To access	VuGen > Tools > Options > Scripting > Output
UI Element	**Description**
Format	**Word wrap** Enables word wrapping in Output pane.
Font	**Text Font** Enables you to select a font. **Size** Enables you to select a font size.

Snapshots

To access	VuGen > Tools > Options > Scripting > Snapshots
UI Element	**Description**

, continued

Enhanced XML view	Enables the following XML viewer visual features: • the XML tree • coloring Clear this option to reduce the amount of memory consumed by the XML view.
Snapshot caching	Enables snapshots to be cached. Clear this option where you are working with large snapshots and are running out of memory.
Do not load text snapshots larger than	Text-based snapshots are not loaded if they are larger than the specified size.
Do not load binary snapshots larger than	Binary [hexadecimal] based snapshots are not loaded if they are larger than the specified size.
Do not load XML snapshots larger than	XML-based snapshots are not loaded if they are larger than the specified size.

Parameters

To access	**VuGen > Tools > Options > Scripting > Parameters**
UI Element	**Description**
Left parameter delimiter	Enables you to specify a left parameter delimiter. The following characters are valid: !, #, $, %, &, (,), [,], {, }, |, ~, `, <, >, ?
Right parameter delimiter	Enables you to specify a right parameter delimiter. The following characters are valid: !, #, $, %, &, (,), [,], {, }, |, ~, `, <, >, ?
Restore Default	Resets both the left and right parameter delimiters to their default values.

Parser

To access	**VuGen > Tools > Options > Scripting > Parser**
UI Element	**Description**

, continued

Enable Language Parser	Disabling the language parser may improve application performance when you are working with very large scripts. However, the following features will be disabled: • Editing step arguments in the **Editor** • Statement completion • Snapshots • Tasks • Thumbnails • Additional step-related functionality This option is enabled by default.

Search and Replace Dialog Boxes

These dialog boxes enable you to find and replace text strings in Vuser scripts and solutions.

Search Dialog Box

To access	• **VuGen > Search > Quick Find** • **VuGen > Search > Find in Files**
Note	Some of the UI elements in the table below appear in the **Quick Find** dialog box only and some appear in the **Find in Files** dialog box only.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Search-type drop-down>	Enables you to specify the type of search to perform: **Quick Find** or **Find in Files**. Note that the UI of the Search dialog box changes depending on the selection.
Find text	Specify the text to search for.
	Shows the Regular Expression Builder. The Regular Expression Builder enables you to build a regular expression in the **Find text** box.
Regular Expression	Select **Regular Expression** to indicate that the **Find text** string is a regular expression.
Scope	Specify the files in which to search.
Match case	Distinguishes between uppercase and lowercase characters during the search.

Match whole word	Searches for occurrences that are whole words, and not part of a larger word.
Search up	Performs the searches upwards.
Include in search	Enables you to select which entities are included in the search.
Directory	Specify the folder that contains the files that will be searched during a "Find in Files" search.
Options	Enables you to specify options when performing a "Find in Files" search.
Find Next	Finds the next occurrence of the text or regular expression in the **Find text** box.
Find All	Finds all occurrences of the text or regular expression in the **Find text** box.

Replace Dialog Box

To access	**VuGen > Search > Quick Replace**

User interface elements are described below:

UI Element	Description
Find text	Specify the text to search for.
	Shows the Regular Expression Builder. The Regular Expression Builder enables you to build a regular expression in the **Find text** box.
Regular Expression	Select **Regular Expression** to indicate that the **Find text** string is a regular expression.
Replace with	Specify the text that will replace the **Find text**.
Scope	Specify the files in which to search.
Options	Enables you to specify options when performing the search.
Match case	Distinguishes between uppercase and lowercase characters during the search.
Match whole word	Searches for occurrences that are whole words, and not part of a larger word.
Search up	Performs the searches upwards.
Find Next	Finds the next occurrence of the text or regular expression in the **Find text** box.
Replace	Replaces the selected text with the text in the **Replace with** box.
Replace All	Replaces all found occurrences of the text or regular expression in the **Find text** box with the text in the **Replace with** box.

Editor

This pane enables you to edit scripts and other related script files. In addition you can open a browser session to search sites, such as the LoadRunner Forum.

To access	The Editor is opened when VuGen is loaded.
Important information	• The Editor is automatically displayed as part of the default layout. • Press Ctlr + Tab to display a list of tabs and panes. Highlight and click to switch tabs in the Editor. • Other main interface panes such as Output, Error, and Snapshot synchronize their displays based on your location in the Editor. • You can double-click an asset in the Solution Explorer to open it in the Editor. • Script modifications are displayed as highlighted text in the editor. The following are examples: ▪ Script has been modified but not saved. ▪ Script has been modified and saved. ▪ Breakpoint has been inserted.
See Also	• "Editor - Overview" on page 51 • " Editor Options" on page 66

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Gutter>	The Editor gutter enables you to add toggle functionality including: • Breakpoints For details, see "Working with Breakpoints" on page 125 or "Breakpoints Pane" on page 96. • Bookmarks For details, see "How to Use Bookmarks" on page 131.
Class/Function Browser	When enabled, you can navigate quickly to a specific class or function in your script by selecting it from the drop down list in the browser.
Line numbers	Display of line numbers in the script.
Context Menu	
Comment region	Enables you to comment out highlighted script lines.
Indent	Indent the selected line or lines of your script.

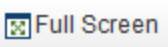
UI Element	Description
Show Snapshot	Show the snapshot associated with the highlighted script step.
Correlate Selection	Opens Design Studio and scans for dynamic values to be correlated in the selected script section.
Insert	 Insert a **Start Transaction** step into your script. For details, see "Transaction Overview" on page 202.  Insert an **End Transaction** step into your script. For details, see "Transaction Overview" on page 202.  Insert a **Rendezvous point** step into your script. For details, see "Rendezvous Points" on page 203.  Insert comments into your script. Log Message Insert a `lr_log_message("enter message here");` into your script.
Breakpoints	Toggle Breakpoint.
Search Community	Opens the default Community Search browser.

Thumbnail Explorer

This pane enables you to flip through thumbnail images of your business process, enhancing your ability to navigate to specific locations in the **Editor** based on a visual representation of a step. Conversely, you can scroll through the **Editor** and see the visual context of your script in the **Thumbnail Explorer**.

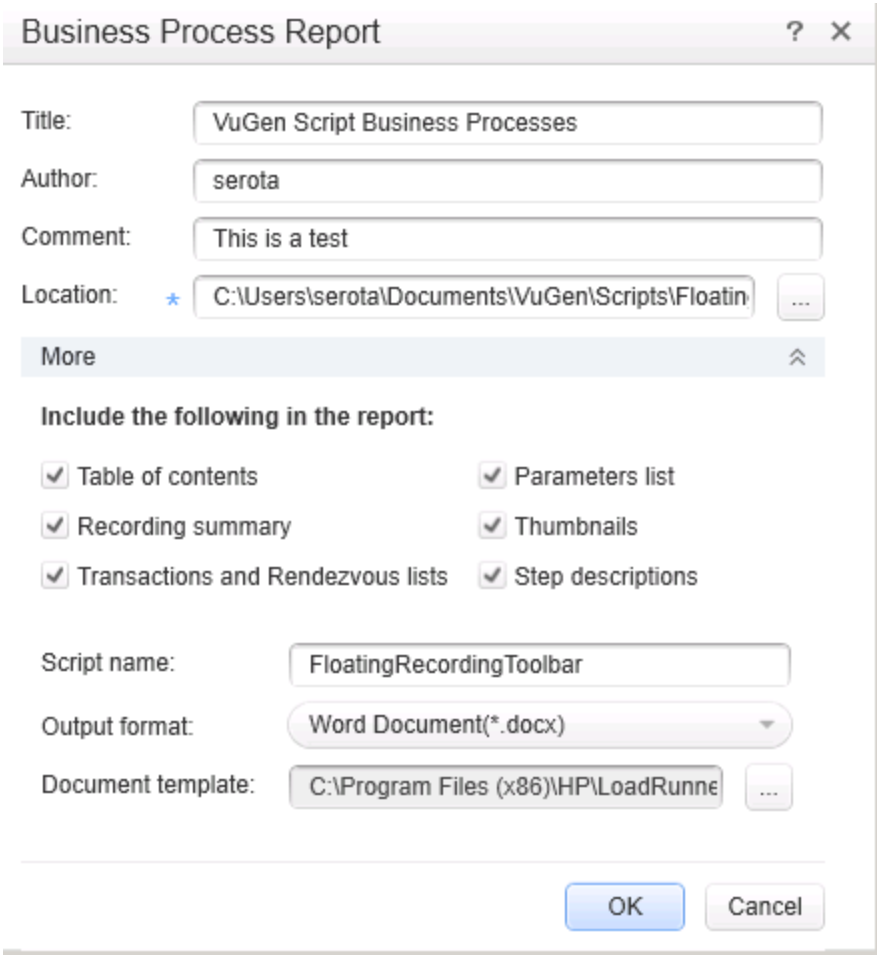
To access	Use one of the following: • **VuGen > View >Thumbnail Explorer** • Click the  button on the VuGen toolbar.

Important information	• You can move this pane to different areas of the Main User Interface. For details see, "VuGen Layouts - Overview" on page 48. • You can configure the **Thumbnail Explorer** in **Options > Scripting Options**. For details, see "Scripting Options" on page 71. • Thumbnails are created in the same order as the actions in the Solution Explorer and not controlled by the settings in **Run Time Settings > General > Run Logic**. • Enabling automatic creation of thumbnails in **Options > Scripting > Thumbnails** enables VuGen to create thumbnails during the application's idle time. • If the Windows Aero theme is enabled, thumbnails capture more realistic images of the application.
Relevant tasks	How to Customize the VuGen Layout

UI Element	Description
	Moves the cursor to the step in the **Editor** associated with the highlighted thumbnail in the **Thumbnail Explorer**.
	Enables a full screen view of the thumbnail.
	Synchronizes the scrolling in the **Editor** with the associated thumbnail in the **Thumbnail Explorer** and step in the **Step Navigator**.
	Filters out minor thumbnails that are not directly related to the recorded business process.
	Refreshes the generated thumbnails.
	Scroll a page left in the **Thumbnail Explorer**.
	Move to the previous thumbnail in the **Thumbnail Explorer**.
	Move to the next thumbnail in the **Thumbnail Explorer**.
	Scroll a page right in the **Thumbnail Explorer**.

Business Process Report Dialog Box

This dialog box enables you to create a business process report.



To access	**VuGen > Tools > Create Business Process Report**
Relevant tasks	"How to Create a Business Process Report" on page 61

User interface elements are described below:

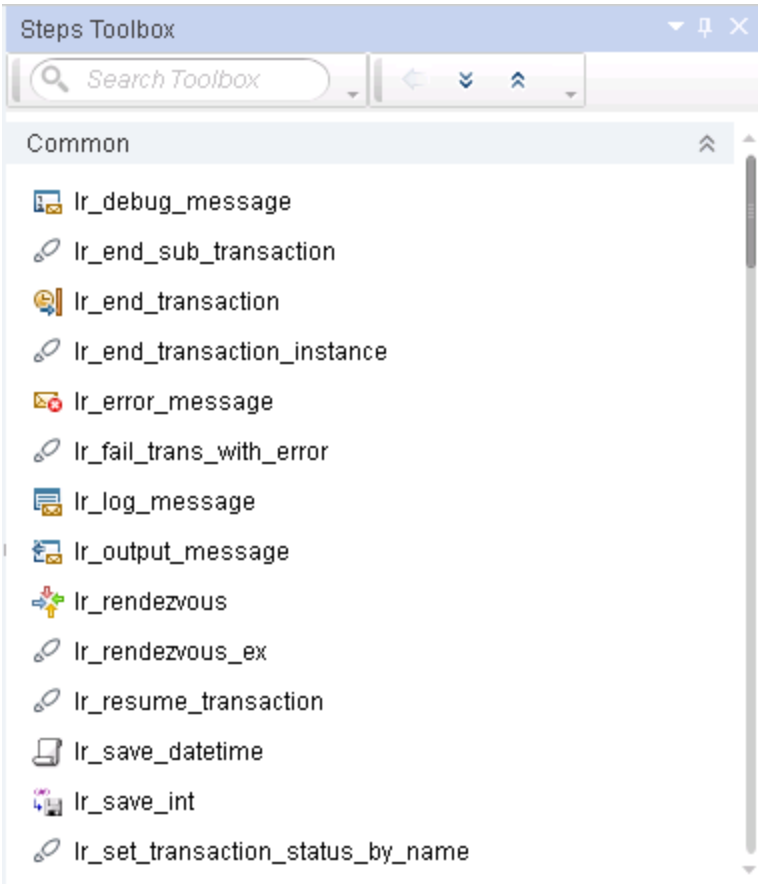
UI Element	Description
Title	The title of the report.
Author	Your name.
Comment	Any additional comments you want to appear in the report.
Location	The path where you want the report to be saved. **Default value:** script folder.
More	Expands the Business Process Report dialog box to display more options.

, continued

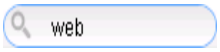
Table of contents	A table of contents generates for your report. If you disable an option, it will not appear in the table of contents. **Default value:** enabled.
Recording summary	A summary of the recording session as it appears when you click the Recording Summary link in the Tasks list. **Default value:** enabled.
Transactions and Rendezvous lists	A list of all of the transactions and rendezvous points that were defined in the script. **Default value:** enabled.
Parameters list	A list of all the parameters that were defined in the script. This list corresponds to the parameters listed in the Parameter List dialog box **Design > Parameter List**. **Default value:** enabled.
Thumbnails	An actual snapshot of the recorded step, adjacent to the step name and description. **Default value:** enabled.
Step descriptions	A short description of each step. **Default value:** enabled.
Script name	The .usr file name of the script.
Output format	Creates the report in the selected format. The following formats are available: • Microsoft Word • Adobe PDF • HTML
Document template	The path and file name of the template to use for the report. The default template is stored in the LoadRunner's **dat** folder. To change the report template, click the browse button and specify new template with a .docx extension. If you want to create a new template, we recommend that you use an existing template as a basis for the new one. This will make sure that the required bookmarks and styles are maintained within the new template.

Steps Toolbox Pane

This pane enables you to drag and drop API functions into your script.

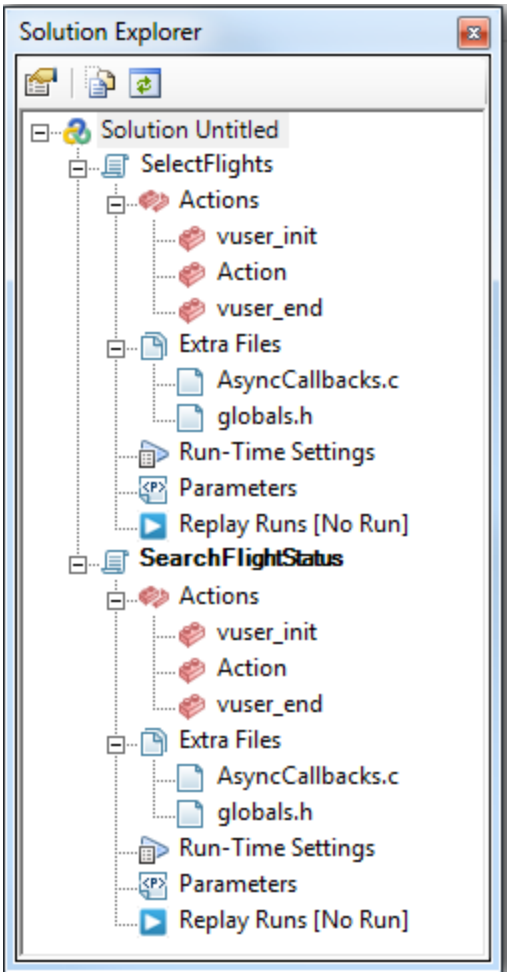
UI example	
To access	Use one of the following: • **Design > Insert in Script > New Step** • Right click in the script in **Editor > Insert > New Step** • Press Ctrl+Alt+B
Important information	• A step's associated parameter dialog box opens when you add the step to the script. • You can drag and drop steps into your script. • You cannot drag and drop a step into a step from the **Steps Toolbox** but you can manually add a step parameter within a step. • If you insert a step into the incorrect location in your script, the script may fail.
Relevant tasks	"How to Insert Steps into a Script" on page 207.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Function List>	Displays a list of available functions divided into the following categories: • Common • Web checks • Services • XML • Web JS • Async
	**Search.** Enables you to perform an incremental search in the function list by entering text. For example, if you type "web" into the search box, the function list will display only those function that include the letters "web".
	**Add.** Add the highlighted step to the current location in your script.
	**Expand/Collapse.** Expand or collapse the step categories.

Solution Explorer Pane

This pane enables you to navigate through a solution which contains script assets, parameters, runtime settings, and replay runs.



To access	Do one of the following: • **View > Solution Explorer** • Press **Ctrl + Alt + L**
Important information	• Solution Explorer is automatically displayed as part of the default layout. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48. • Other main interface panes such as Output, Error and Snapshot synchronize their displays based on your location in the Solution Explorer. • You can double-click an asset to activate it in the editor area or right-click to examine quick operations available for that asset.
See also	• "VuGen Main User Interface - Overview" on page 41 • "Solution Explorer - Overview" on page 50 • "VuGen Layouts - Overview" on page 48 • "How to Import Actions to a Script" on page 113

UI Element	Description
<Solution>	Container for scripts. • Default solution name is Untitled. Context menu options • Add New Script For details, see "Create a New Script Dialog Box" on page 120. • Add Existing Script • Save All Scripts • Close Solution • Save Solution As...

<Script>	Container for script assets including scripts actions, extra files, run-time settings and parameters. Context menu options • **Save Script** Select to save script • **Save Script As...** Save script with new name or to a new location, such as ALM. • **Export to Template....** You can save scripts as templates. For details, see "How to Create and Open Vuser Script Templates" on page 111. • **Remove Script** Enables you to delete a script from the solution or select checkbox to delete from file directory. • **Select file to compare** You can compare one asset to another. This option selects the highlighted asset as the primary asset. • **Compare to <filename>** Compares between the selected asset and the primary asset. • **Compare to external file...** You can compare an asset to a file outside of the solution. This option sets the highlighted asset as the primary asset and opens Windows Explorer to enable you to select the secondary asset. • **Select Folder to Compare** You can compare one folder to another. This option selects the highlighted folder as the primary folder. • **Compare to External Folder...** Compares between the selected folder and the primary folder. • **Open Script Folder** While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script. You can open the folder on the local disk where the script is saved. For scripts that are saved on a different storage location (such as ALM), this option opens the temporary folder on the local disk.

<Actions>	Container for individual actions including the default actions: • vuser_int • action • vuser_end Context menu options • Create New Action Enables you to add a new script action block to your script. • Import Action Enables you to import an action from an existing script.
Extra Files	Container for extra files associated with your script. Context menu options • Add files to script Enables you to add extra files to the Extra Files node of your script. • Add files from folders and sub-folders Enables you to add the contents from folders or sub-folder to the Extra Files node of your script.
Run-Time Setting	Opens the run-time settings dialog box. For details, see "Run-Time Settings Overview" on page 332.
Parameters	Enables you to create, edit, and list parameters associated with your script. For details, see "Parameters" on page 227. Context menu options • Parameters list • Create new parameter • Edit parameter • Configure parameter delimiter
Replay Runs	Enables you to display the Replay Summary for a selected iteration. Context menu options • <Select run iteration> • Open Replay Summary Enables you to open the Replay Summary in the Editor for selected iteration. • Open Test Results Enables you to open the Test Results in the Editor for selected iteration.

Output Pane

The Output pane displays messages that were generated during the recording, compilation, and replay of your script.

To access	Select **View > Output**, or click the **Show Output Pane** button on the VuGen toolbar.
Important information	You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41

User interface elements are described below:

UI Element	Description
	The type of output to display. The following types are available: • **Replay.** Displays the messages generated by the script replay. Note that if you double-click an entry in the Replay log, VuGen moves the cursor to the corresponding line in the Editor. • **Compilation.** Displays the compilation messages. • **Code Generation.** Displays the code generated during the recording. • **Recording.** Displays the messages generated during the recording. • **Recorded Events.** Displays events that occurred during recording.
	Clears all of the messages from the message list.
	**Toggle Line Wrap.** When selected, wraps the text of each message onto the next line - as required.
	Jumps to the location in the source document relevant to the selected output message.

UI Element	Description
	**<Find box>.** The text string that you want to find. You can refine your search by selecting one of the **Options** described below. Press ENTER to begin the search.
	**Find Previous / Find Next.** Highlights the next or previous string that matches the text you entered in the **Find** box. These buttons are available only after you enter text in the **Find** box.
	Enables you to refine your search with the following options: • **Match Case.** Distinguishes between upper-case and lower-case characters in the search. • **Match Whole Word.** Searches for occurrences that are only whole words and not part of longer words. • **Use Regular Expression.** Treats the specified text string as a regular expression. **Note:** Extended regular expressions and multi-line searches are not supported.
	Opens the Save As dialog box, enabling you to save the contents of the message list as a text file.
View Summary [Available in the Replay log only]	Opens the Replay Summary tab. For details, see "Replay Summary Tab" on page 100.

Snapshot Pane

To access	Select **View > Snapshot**, or click the **Show Snapshot Pane** button  on the VuGen toolbar.
Important information	• The appearance and functionality of the Snapshot pane vary depending on the protocol of the current Vuser script. In addition to the standard controls, the Snapshot pane may display controls that are specific to the current Vuser protocol. • You can move this pane to different areas of the Main User Interface. For details see, "VuGen Layouts - Overview" on page 48.
See also	"Snapshot Pane - Overview" on page 54 "How to Work with Snapshots" on page 57

Standard Snapshot pane controls

The Snapshot pane displays the following standard controls:

UI Element	Description
Split	Splits the Snapshot pane to show two snapshots.
Horizontal	Displays two snapshots in the Snapshot pane - one to the side of the other.
Vertical	Displays two snapshots in the Snapshot pane - one above the other.
Single	Shows a single snapshot in the Snapshot pane.
Recording	Shows the record snapshot in the Snapshot pane.
Replay	Shows the replay snapshot in the Snapshot pane.
Iteration: 1	Select the iteration number of the replay snapshot to display.
	Synchronizes the display of the two snapshots if the Snapshot pane is split. Note that snapshot synchronization is available for only specific Vuser protocols, and for only specific views within the protocols.
Compare	Compares the two snapshots that are currently displayed in the Snapshot pane. To enable the **Compare** functionality, you must first split the Snapshot pane to show two snapshots. By default, VuGen uses the *WDiff* utility to compare snapshots. You can specify an alternative comparison tool as described in "Scripting Options" on page 71. Note that the snapshot comparison functionality is available for only the Web HTTP/HTML and Web Services protocols.

Citrix, RDP, and SAP protocols

User interface elements are described below:

UI Element	Description
	Displays the next record snapshot that was appended during script replay.
	Displays the previous record snapshot that was appended during script replay.
Image	Displays a graphical representation of the snapshot. You can synchronize the display of two snapshots in the Snapshot pane. Snapshots display faster when the Image view is used than when the Full view is used.
Full	Displays a graphical representation of the snapshot. You cannot synchronize the display of two snapshots in the Snapshot pane. Snapshots display slower when the Full view is used than when the Image view is used.

Windows Sockets protocol

User interface elements are described below:

UI Element	Description
Hex	Displays the buffer data in hexadecimal.
Text	Displays the buffer data as text.
Go To	Open the Go To Offset dialog box that enables you to navigate within the data buffer.

Ajax (Click & Script) and Web (Click & Script) protocols

User interface elements are described below:

UI Element	Description
Page View	Displays a graphic view of the Web page.
Page Source	Displays the HTTP source code of the Web page.

Web (HTTP/HTML) protocol

User interface elements are described below:

UI Element	Description
	Displays step data in HTML format.
	Displays step data in HTTP format. This enables you to view in-depth information about the step, including request data, response data, cookies, and headers. For more information, see "Web Snapshots - Overview" on page 676.
	(HTTP Data view only) Displays the HTTP flow data in a list format.
	(HTTP Data view only) Displays the HTTP flow data in a tree structure.

Bookmarks Pane

The Bookmarks pane displays a list of the bookmarks in your Vuser script. You can navigate between the bookmarks to help analyze and debug your code.

To access	**VuGen > View > Bookmarks**

, continued

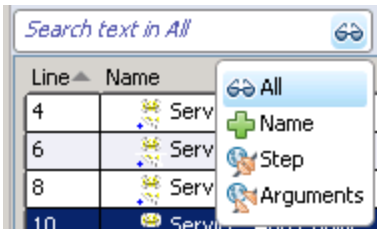
Important information	• All bookmarks added to a Vuser script are maintained after you close and reopen the Bookmarks pane. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	"How to Use Bookmarks" on page 131 How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Bookmarks list>	Displays a list of all bookmarks that are defined in the Vuser script. You can double-click any bookmark line to navigate directly to the relevant line in the Vuser script.
	Toggles the status of the selected bookmark.
	Navigates to previous bookmark in the pane.
	Navigates to next bookmark in the pane.
	Deletes the selected bookmark.
	Deletes all bookmarks.

Step Navigator Pane

The Step Navigator pane enables you to navigate to a selected step in your script. If your script contains many steps, you can use the search box to search for matching text in the different parts of the steps.



To access	**VuGen > View > Steps**

, continued

Important information	• You can search within an action or script but not across multiple scripts in a solution. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.

User interface elements are described below:

UI Element	Description
Search box	You can search different parts of the steps for matching text. The following parts of the steps can be searched: • **All**. (Default) All parts of the step (name, step, arguments). • **Name**. The name of the step. • **Step**. The description of the step (for example, web_url, web_custom_request) • **Arguments**. The arguments for the step. Enter the text you want to search for in your steps, in the search edit box and select the part of the steps you want to search. The Steps pane displays only those steps that match your search criteria.
Line	The number of the step in the script.
Name	A step name.
Step	The step type.
	**File parser indicator.** A green symbol indicates that parsing succeeded and a red symbol indicates that parsing failed.
Action	The action into which the step was created.
# steps displayed	Displays the total number of steps in the script or in the action.

Errors Pane

The Errors pane lists the replay and syntax errors found in your script, and enables you to locate each error so that you can resolve it.

To access	**VuGen > View > Errors**
Important information	You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Define Available Categories>	Filters the error list by the source of the error.
Errors: 0	Shows or hides syntax errors.
Warnings: 0	Shows or hides warnings detected during the run.
Messages: 0	Shows or hides informational messages detected during the run.
Locate	Jumps to the line in the Editor that contains the error.
! <Exclamation Point>	Message type: • Error • Warning • Informational message
Line	The line containing the error.
Description	Description of the error, warning or message and advice on how to fix the problem. For example, a syntax error is displayed if you opened a conditional block with an If statement but did not close it with an End If statement, the description is Expected Expression. Note: If the description does not fit within the Description column, a tooltip displays the full description when you hover the cursor over the column. In certain cases, VuGen is unable to identify the exact error and displays a number of possible error conditions, for example: Expected 'End Sub', or 'End Function', or 'End Property'. Check the statement at the specified line to clarify which error is relevant in your case.
File	The name of the file that contains the problematic statement.
Path	The full path of the file that generated the error.
Testing Project	The name of the relevant script.

Task Pane

This pane enables you to add, edit and track tasks associated with an individual script or the overall goals of the project.

To access	**VuGen > View > Task Pane**

, continued

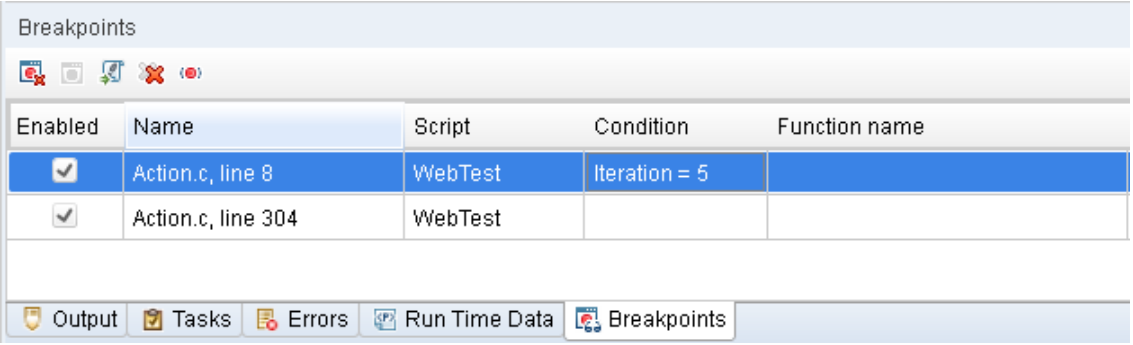
Important information	You can move this pane to different areas of the Main User Interface. For details see, "VuGen Layouts - Overview" on page 48.
Relevant tasks	How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Comments tab	Comment tasks are added directly into your script using the comment syntax of your scripting language and include a keyword such as TODO or FIXME. For example, in C a comment script would look like this: `//TODO Add Parameter` The Task Pane displays the following information about each task: • !: Task Type • Line: What line the task is located. Double-clicking the task jumps to that location in the script. • Description: The Keyword and the task contents. • File: Action • Path: File location of the action.
User tab	You can add, edit, delete user tasks: Click  to add a task . Click  to edit a task . Click  to delete a task.
<Task Filter>	You can filter tasks associated with a particular script or see all the tasks associated with the solution.

Breakpoints Pane

The Breakpoints pane enables you to set and manage breakpoints to help analyze the effects of the script on your application at pre-determined points during script execution.



To access	**VuGen > View > Debug > Breakpoints**
Important information	You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	"How to Debug Scripts with Breakpoints" on page 130

Breakpoints Pane

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Removes the selected breakpoint.
	Removes all breakpoints.
	Locates the cursor in the Vuser script at the line that contains the selected breakpoint.
	Disables the selected breakpoint if it is enabled, and enables the selected breakpoint if it is disabled.
	Allows you to enter conditions for the selected breakpoint. See "Breakpoint Condition Dialog Box" on next page for more details.
<Breakpoints Grid>	A list of the breakpoints and their locations in the script. To enable a breakpoint, select the **Enable** check box next to that breakpoint. To disable a breakpoint, clear the **Enable** check box.
Enabled	A checkbox that specifies whether the breakpoint is enabled or disabled, and enables you to enable or disable the adjacent breakpoint.
Name	The name of the file that contains the breakpoint, and the line number within the file that contains the breakpoint.
Script	The name of the Vuser script that contains the breakpoint.
Condition	The condition that applies to this breakpoint. If there is no condition, the replay will always stop at the breakpoint.

, continued

Function Name	The name of the function within the Vuser script that contains the breakpoint.

Breakpoint Condition Dialog Box

The Breakpoint Condition dialog box enables you to condition breakpoints by the iteration number, the value of one or more parameters, or a combination of both parameter values and iteration number.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Break when iteration number is	A checkbox that enables you to specify a breakpoint dependant on the iteration number.
<Operand>	Choose one of the following operands: • = (equal) • <= (less than or equal to) • => (equal to or greater than) • < (less than) • > (greater than)
to	Enter an iteration number.
Break when Parameter	A checkbox that enables you to specify a breakpoint dependant on the value of a parameter.
<Parameter Name>	Choose a parameter from the drop down list
Value is	Enter the parameter value for which you want a breakpoint.
OK	Apply the conditions to the selected breakpoint. All future replays will only stop at this breakpoint if these conditions are met.

Watch Pane

The Watch pane enables you to monitor variables while a script runs.

To access	**VuGen > View > Debug > Watch**
Important information	• This pane is relevant only when a run session is paused. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.

User interface elements are described below:

UI Element	Description
	Enables you to add a variable to the watch list.
	Enables you to edit the selected variable in the watch list.
	Deletes the selected variable from the watch list.
	Deletes all the variables from the watch list.
Expression	The variable whose value you want to watch.
Value	The current value of the variable. The evaluated value is displayed only when a run session is paused.
Type name	The type of the variable's value after it is evaluated (for example, **Integer** or **Char**). If an variable cannot be evaluated in the current context, the type displayed is **Incorrrect expression**.

Run Time Data Pane

The Run Time Data pane displays information about the current script execution.

To access	**VuGen > View > Debug > Run Time Data**
Important information	• The Run Time Data pane is accessible during script replay only. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41 "How to Replay a Vuser Script" on page 127

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Iteration	Displays the current iteration number.
Action	Displays the Action name of the currently replayed step.
Line Number	Displays the line number of the currently replayed step.
Elapsed time	Displays the time that has elapsed since the start of the replay.
<Parameters>	Displays all parameters defined, together with the script and their substitution values based on the selected update method (sequential, unique, etc.). VuGen shows this information even if the parameter is not used in the script.

Call Stack Pane

This debug pane enables you to view information about the functions that are currently on the call stack of your script.

To access	**VuGen > View > Debug> Call Stack**
Important information	• This pane is relevant only when a run session is paused. • You can double-click any element in the Call Stack pane to navigate to the beginning of the relevant function/action. • This pane is read-only. • You can move this pane to different areas of the Main User Interface. For details, see "VuGen Layouts - Overview" on page 48.
Relevant tasks	How to Customize the VuGen Layout
See also	"VuGen Main User Interface - Overview" on page 41

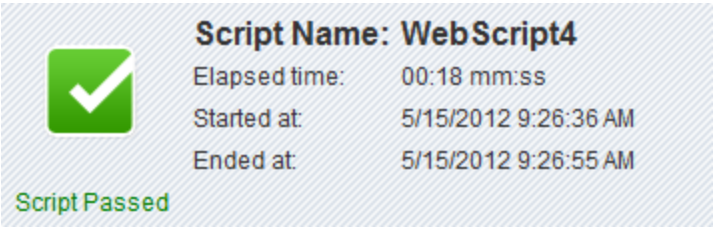
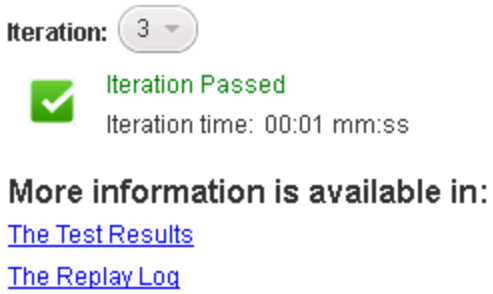
User interface elements are described below:

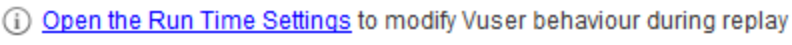
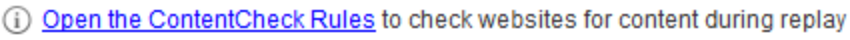
UI Element	Description
Function name	The name of the function currently called.
File name	The name of the file containing the called function.
Line #	The line number on which the function definition begins.

Replay Summary Tab

This tab provides summarized replay results and links to script replay details.

To access	Use one of the following: • **Solution Explorer >** Right click **Replay Runs >** Select **Open Replay Summary Tab** • Select View Summary link on the **Output Pane**
See Also	• "Output Pane" on page 89 • "How to Replay a Vuser Script" on page 127

UI Element	Description
<Script Dashboard> 	Displays basic script information including: • Script name • Replay Status • Elapsed time • Start time • End time
<Results Dashboard> 	Provides links for replay result details: • **Iteration** Enables you to select a specific iteration result set from a drop down. Displays the iteration run status and iteration time. • **Test Results** For details, see "Test Results Window" on page 216. • **Replay Log**

UI Element	Description
<Replay Button> 	Enables you to replay your script from the **Replay Summary Tab.**
<Modify RunTime Settings> 	Enables you to access **Run Time Settings** for your active script. For details, see "Run-Time Settings Overview" on page 332.
<Modify Content Check Rules> 	VuGen's **ContentCheck** mechanism enables you to detect all types of errors sent by the web server. For details, see "Internet Protocol > ContentCheck Node" on page 362.

VuGen - Troubleshooting and Limitations

This section describes troubleshooting and limitations for the VuGen main user interface.

Troubleshooting Snapshots

If you encounter a step without a snapshot, follow these guidelines to determine why it is not available. Note that not all steps are associated with snapshots—only steps with screen operations or for Web, showing browser window content, have snapshots.

Several protocols allow you to disable the capturing of snapshots during recording using the Recording options.

If there is no **Record** snapshot displayed for the selected step, it may be due to one of the following reasons:

- The script was recorded with a VuGen version 6.02 or earlier.
- Snapshots are not generated for certain types of steps.
- The imported actions do not contain snapshots.

If there is no **Replay** snapshot displayed for the selected step, it may be due to one of the following reasons:

- The script was recorded with VuGen version 6.02 or earlier.
- The imported actions do not contain snapshots.
- The Vuser files are stored in a read-only folder, and VuGen could not save the replay snapshots.
- The step represents navigation to a resource.

Protocol Advisor Overview

You use the Protocol Advisor to help you determine an appropriate protocol for recording a Vuser script. The Protocol Advisor scans your application for elements of different protocols and displays a list of the detected protocols. These protocols should be used as guidelines and as a starting point for finding the optimal protocol for your application.

In most cases, more than one protocol is suggested and displayed, along with combinations of protocols. For guidelines on how to use the list of suggested protocols, see "How to use the Protocol Advisor" below.

How to use the Protocol Advisor

This task describes a typical workflow for finding the optimal protocol to record your application using the Protocol Advisor.

1. **Start the Protocol Advisor**

   In VuGen, select **Record > Protocol Advisor > Analyze Application**

   If you are analyzing a web application, you will need to specify:

   - Program to analyze
   - URL Address
   - Working Directory

   If you are analyzing a Windows application, you will need to specify:

   - Program to analyze
   - Program arguments, if any
   - Working Directory

   For details, see "Protocol Advisor Dialog Box" on page 105.

2. **Perform typical business processes**

   Perform typical business processes in your application. Try to walk through a variety of business processes to make sure that your results are comprehensive. Click **Stop Analyzing** to end the analysis and display the results.

3. **Save the results**

   Select **Record > Protocol Advisor > Save Results** or click **Save Results** from the toolbar at

the top of the report. Enter a name and select the folder.

To display a previous report, select **Record > Protocol Advisor > Recent reports**. VuGen will maintain a history of the last sixteen Protocol Advisor reports generated. To clear the history, select **Record > Protocol Advisor > Clear Recent Reports**.

4. ## Select a protocol and create a new Vuser script

   Select one of the protocols using the following order of priority and create a Vuser script using that protocol:

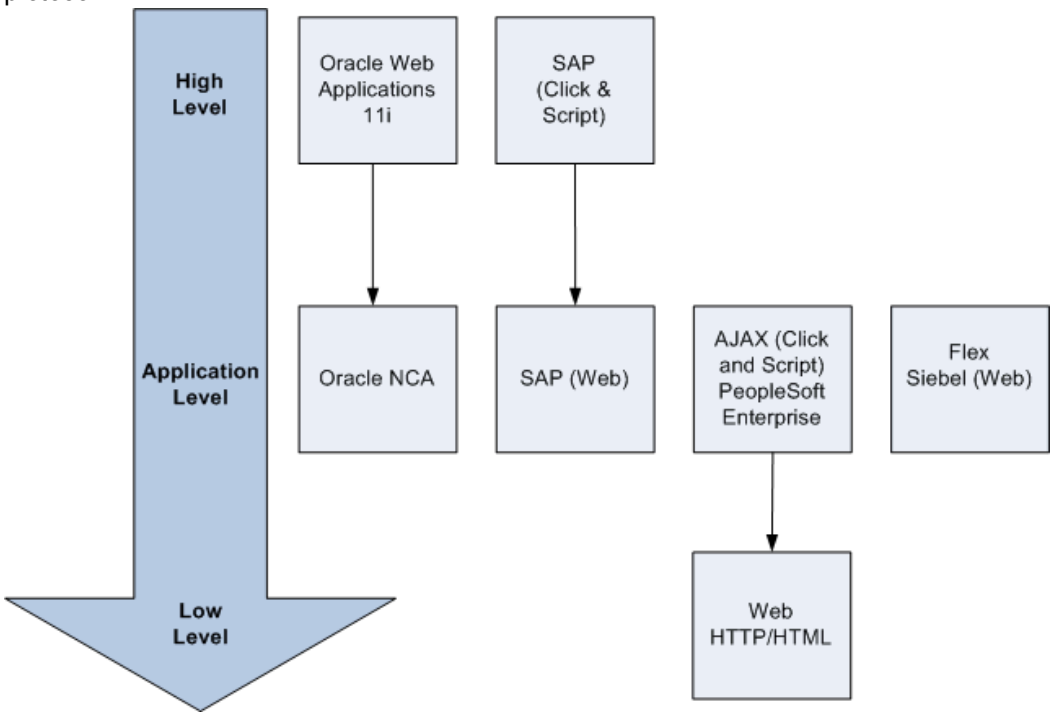
   - Multi/Combination protocol
   - The highest level application protocol
   - The first protocol on the list

5. ## Enhance, debug, and verify replay

   Enhance and debug your script until you can replay it successfully. If, after enhancing and debugging the Vuser script, you cannot replay it successfully, proceed to the next step.

6. ## If unsuccessful replay, select a different protocol and repeat

   - **For all non Web-based protocols:** Return to the Protocol Advisor Results page and select the next protocol on the list and repeat previous steps. If there are no more protocols listed, contact HP Customer Support.
   - **For Web-based protocols:** The following diagram illustrates the Web-based protocols arranged according to their application level. The arrows indicate the order in which you should attempt a new protocol. For example, if the first protocol you used was Oracle Web Applications 11i and it failed to successfully replay, you would then try the Oracle NCA protocol.



Protocol Advisor Dialog Box

This dialog box enables you to analyze your application to determine the appropriate VuGen protocols to use for recording.

To access	Do one of the following: • **VuGen > Record > Protocol Advisor > Analyze Application** • **VuGen > File > New Script and Solution > Protocol Advisor**
Important information	The options in this dialog box change according to the selection you make for **Application type**.
Relevant tasks	"How to use the Protocol Advisor" on page 103

User interface elements are described below:

UI Element	Description
Application type	Windows or Web application.
Program arguments (Win32 Applications only)	Specify command line arguments for the executable specified above. For example, if you specify *plus32.exe* with the command line options `peter@neptune`, it connects the user *Peter* to the server *Neptune* when starting *plus32.exe*. This option is displayed only for Windows applications.
Program to analyze	Select the browser, Web application, or Windows application to analyze.
URL Address (Internet Applications only)	The starting URL address. This option is displayed only for Web applications.
Working directory	The working directory for your application, if needed.

Protocol Advisor - Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Protocol Advisor.

- This feature will only detect protocols supported by LoadRunner.
- Some Web protocols are detected based on the URL. For example, the URL may contain keywords such as SAP. Therefore, if you use a different URL or a different application with the same URL, the results may differ.
- The following protocols are frequently detected but are not always appropriate for use. You

should only use them if there are no other detected protocols.

- COM/DCOM
- Java
- .NET
- WinSock
- LDAP

Recording

Vuser Script Templates

The User-Defined Template enables you to save a script with a specific configuration as a template. You can then use this template as a basis for creating future scripts.

The template supports the following files and data:

- Run-Time Settings
- Parameters
- Extra files
- Actions
- Snapshots

Recording and General options are not supported as they are generic settings and are not relevant to a specific script.

Notes and Limitations

- Once you have configured a script for a specific protocol and then save the script as a template, further scripts based on that template will only work with that same protocol. For example, if you configured your script for the Web (Click & Script) protocol, then created a template from that script, the template can only be used with Web (Click & Script).
- Once you have created your template, you cannot edit it directly in VuGen. To make any changes, you open the template and save it again with another name or overwrite the existing template.
- If you regenerate an original script from a template, you will lose all of your manual changes.

Script Sections

Each Vuser script contains at least three sections: vuser_init, one or more Actions, and vuser_end. Before and during recording, you can select the section of the script into which VuGen will insert the recorded functions. The following table shows what to record into each section, and when each section is executed:

Script Section	Used when recording...	Is executed when...
vuser_init	a login to a server	the Vuser is initialized (loaded)
Actions	client activity	the Vuser is in **Running** status
vuser_end	a logoff procedure	the Vuser finishes or is stopped

When you run multiple iterations of a Vuser script, only the *Actions* sections of the script are repeated—the *vuser_init* and *vuser_end* sections are not repeated. For more information on the iteration settings, see "General > Run Logic Node" on page 361.

You use the VuGen script editor to display and edit the contents of each of the script sections. You can display the contents of only a single section at a time. To display a section, highlight its name in the left pane.

When working with Vuser scripts that use Java classes, you place all your code in the Actions class. The Actions class contains three methods: init, action, and end. These methods correspond to the sections of scripts developed using other protocols—you insert initialization routines into the init method, client actions into the action method, and log off procedures in the end method.

For more information, see "Java Protocol - Manually Programming Scripts" on page 544.

```
public class Actions{
    public int init() {
        return 0;}
    public int action() {
        return 0;}
    public int end() {
        return 0;}
}
```

Note: Transaction Breakdown for Oracle DB is not available for actions recorded in the vuser_ init

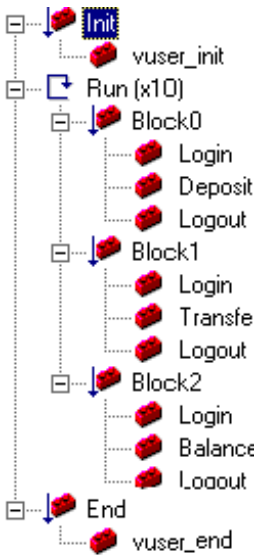
Script Section Structure Example

Every Vuser script contains three sections: *vuser_init*, *Run (Actions)*, and *vuser_end*. You can instruct a Vuser to repeat the *Run* section when you run the script. Each repetition is known as an *iteration*.

The *vuser_init* and *vuser_end* sections of a Vuser script are not repeated when you run multiple iterations.

When you run scripts with multiple actions, you can indicate how to execute the actions, and how the Vuser executes them:

In the following example, *Block0* performs a deposit, *Block1* performs a transfer, and *Block2* submits a balance request. The *Login* and *Logout* actions are common to the three blocks.



Sequence. You can set the order of actions within your script. You can also indicate whether to perform actions sequentially or randomly.

Iterations. In addition to setting the number of iterations for the entire *Run* section, you can set iterations for individual actions or action blocks. This is useful, for example, in emulating a commercial site where you perform many queries to locate a product, but only one purchase.

Weighting. For action blocks running their actions randomly, you can set the *weight* or percentage of each action within a block.

In most cases, the name of the header file corresponds to the prefix of the protocol. For example, Database functions that begin with an **lrd** prefix, are listed in the **lrd.h** file.

Header Files

Header files commonly contain forward declarations of classes, subroutines, variables, and other identifiers. In most cases, the name of the header file corresponds to the prefix of the protocol. For example, Database functions that begin with an **lrd** prefix, are listed in the **lrd.h** file.

The following table lists the header files associated with the most commonly used protocols:

Protocol	File
Ajax (Click and Script)	**web_ajax.h**
Citrix	**ctrxfuncs.h**
COM/DCOM	**lrc.h**
Database	**lrd.h**
FTP	**mic_ftp.h**
General C function	**lrun.h**
IMAP	**mic_imap.h**

LDAP	**mic_mldap.h**
MAPI	**mic_mapi.h**
Oracle NCA	**orafuncs.h**
POP3	**mic_pop3.h**
RDP	**lrrdp.h**
SAP GUI	**as_sapgui.h**
SAP (Click and Script)	**sap_api.h**
Siebel	**lrdsiebel.h**
SMTP	**mic_smtp.h**
Terminal Emulator	**lrrte.h**
WAP	**as_wap.h**
Web (HTML\HTTP)	**as_web.h**
Web (Click and Script)	**web_api.h**
Web Services	**wssoap.h**
Windows Sockets	**lrs.h**

Script Directory Files

While you record, VuGen creates a series of configuration, data, and source code files. These files contain Vuser run-time and setup information. VuGen saves these files together with the script. To open the script folder, right click on the script name in **Solution Explorer** and select **Open Script Folder**.

Multiple Protocol Scripts

When you record a single protocol, VuGen records only the specified protocol. When you record in multi-protocol mode, VuGen records the actions in several protocols. Multi-protocol scripts are supported for the following protocols: COM, FTP, IMAP, Oracle NCA, POP3, Window Sockets (raw), SMTP, and Web.

Another variation between Vuser types is multiple-action support. Most protocols support more than one action section. Currently, the following protocols support multi-actions: Oracle NCA, Web, RTE, General (C Vusers), WAP, and VoiceXML.

- For most Vuser types, you create a new Vuser script each time you record—you cannot record into an existing script. However, when recording a Java, Web, WAP, Oracle NCA, or RTE Vuser script, you can also record within an existing script.
  a.

Since VuGen supports a large variety of protocols, some of the recording steps that follow apply only to specific protocols.

For all Java language Vusers (CORBA, RMI, Jacada, and EJB) see "Java Protocol" on page 529.

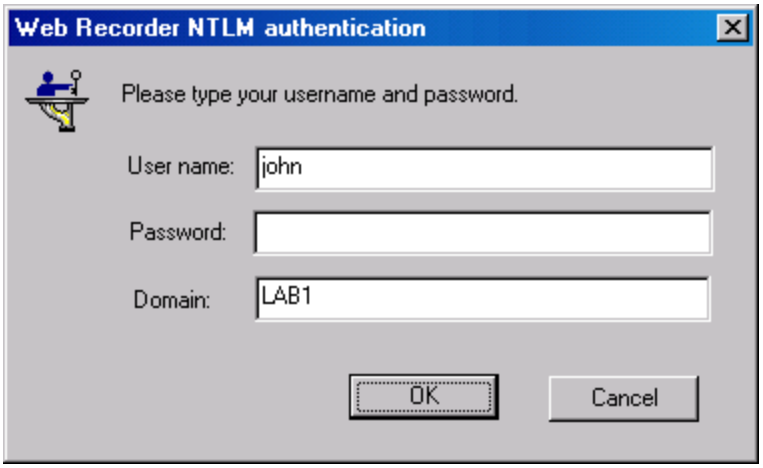
Providing Authentication Information

The following section only applies to multi-protocol scripts.

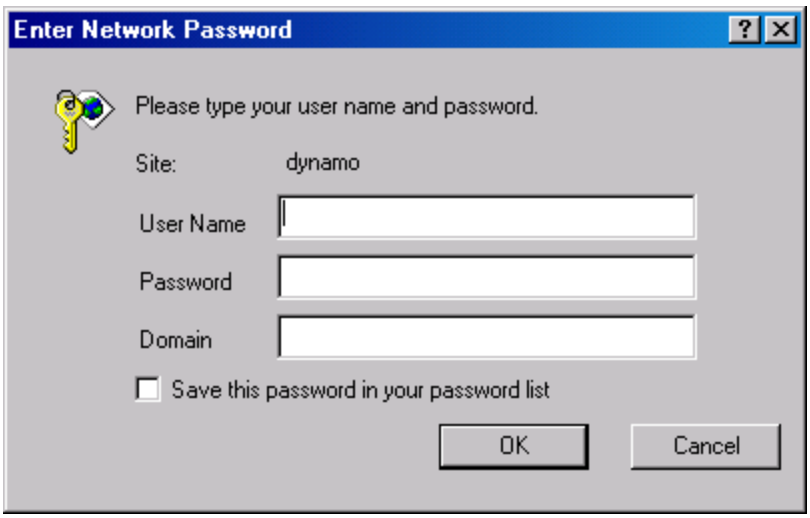
When recording a Web session that uses NTLM authentication, your server may require you to enter details such as a user name and password.

Initially, IE (Internet Explorer) tries to use the NT authentication information of the current user:

- If IE succeeds in logging in using this information and you record a script —then, at the end of the recording VuGen prompts you to enter a password. VuGen retrieves the user name and domain information automatically. If necessary, you can also edit the user name in the Web Recorder NTLM authentication dialog box.



- If IE is unable to log in with the current user's information, it prompts you to enter a user name and password using the standard browser authentication dialog box.



Generating a web_set_user function

When performing NTLM authentication, VuGen adds a **web_set_user** function to the script.

- If the authentication succeeds, VuGen generates a **web_set_user** function with your user name, encrypted password, and host.

```
web_set_user("domain1\\dashwood",
          lr_decrypt("4042e3e7c8bbbcfde0f737f91f"),
         "sussex:8080");
```

- If you cancel the Web Recorder NTLM Authentication dialog box without entering information, VuGen generates a **web_set_user** function for you to edit manually.

```
web_set_user("domain1\\dashwood,
        "Enter NTLM Password Here",
        "sussex:8080");
```

Note: If you enter a password manually, it will appear in the script as-is, presenting a security issue. To encrypt the password, right-click the password and select **Encrypt string**. VuGen encrypts the string and generates an **lr_decrypt** function, used to decode the password during replay. For more information about encrypting strings, see "Encrypting Text" on page 202.

How to Record a Vuser Script

This task describes how to record a Vuser script.

1. **Configure the recording options - optional**

   The recording options contain many different options that affect your script during the recording and generation stages of creating a script. For concept and user interface details, see "Recording Options" on page 258.

2. **Initialize the recording session**

   When creating a new script, this occurs automatically. To start recording manually, click the **Start Record** button on the VuGen toolbar and complete the Start Recording dialog box. For user interface details, see "Record Dialog Box" on page 114.

3. **Perform a business processes on your application**

   VuGen will automatically open your application as well as a floating toolbar and begin recording your actions. Perform the desired business processes that you wish to record. The toolbar allows you to insert transactions, rendezvous points, comments, as well as determine which section of the script to record in while recording your application. For user interface details, "VuGen Main User Interface - Overview" on page 41.

   Click the **Stop** button on the floating toolbar when you are finished recording.

How to Create and Open Vuser Script Templates

This task describes how to create, create from, and rename Vuser script templates.

Create a Vuser Script Template

Open a script in VuGen, select **File > User-Defined Templates > Export to Template** and enter a name for the template.

Create a Vuser Script From a Template

Select **File > New Script and Solution > VuGen > User Templates** and select the template file.

Rename a Template

1. Select **File > User-Defined Templates > Manage from Explorer**
2. In the Explorer dialog box:
   a. Rename the content file (.zip)
   b. Rename the description file (.xpt)
3. Using a text editor, modify the following tags in the .xpt file:
   a. **Name tag**: `<Name>NewName </Name>`
   b. **File tag src property:** `<File name="template_temp.zip" src="NewName.zip" binary="True" />`

Notes and Limitations

- Once you have configured a script for a specific protocol and then saved the script as a template, further scripts based on that template will only work with that same protocol. For example, if you configured your script for the Web (Click & Script) protocol, then created a template from that script, the template can only be used with Web (Click & Script).
- Once you have created your template, you cannot edit it directly in VuGen. To make any changes, you open the template and save it again with another name or overwrite the existing template.

How to Work with .zip Files

VuGen allows you to work with .zip file in several ways. The advantages of working with .zip files is that you conserve disk space, and it allows your scripts to be portable. Instead of copying many files from machine to machine, you only need to copy one .zip file.

Import from a Zip File

To open a script stored in a .zip file, select **File > Manage Zip Files > Import from Zip File**. After you select a .zip file, VuGen prompts you for a location at which to store the unzipped files.

Export to Zip File

To save the entire script folder as a .zip file, select **File > Manage Zip Files > Export to Zip File**.

You can indicate whether to save all files or only runtime and specify the compression ratio.

Zip and Email

To create a .zip file and send it as an email attachment, select **File > Manage Zip Files > Zip and Email**. When you click **OK** in the **Zip and Email** dialog box, VuGen compresses the file according

to your settings and opens an email compose form with the .zip file as an attachment.

Edit Script in Zip File

To work from a .zip file, while not expanding or saving the script files, select **File > Manage Zip Files > Edit Script in Zip File...**. When you modify the script and save it, the changes are stored directly in the .zip file.

How to Import Actions to a Script

For Vuser types that support multiple actions, you can import actions into your script from another Vuser script. You can only import actions from Vusers of the same type. Note that any parameters associated with the imported action, will be merged with the script. The following steps describe how to import actions into the current script.

1. Select **Design > Action > Import Action**, or right-click the Solution Explorer and select **Import Action** from the right-click menu. The Import Action into VuGen Script dialog box opens.
2. Click **Browse** to select a Vuser script. A list of the script's actions appears in the **Actions to Import** section.
3. Select the actions you want to include and click **Import**. The imported action(s) are displayed in the **Solution Explorer**.

How to Regenerate a Vuser Script

If you need to revert back to your originally recorded script, you can regenerate it. This feature is ideal for debugging, or fixing a corrupted script. When you regenerate a script, VuGen removes all of the manually added enhancements to the recorded actions. If you added parameters to your script, VuGen restores the original values. The parameter list, however, is not deleted; you can reinsert parameters that you created earlier. Note that regeneration only cleans up the recorded actions, but not those that were manually added.

This task describes how to regenerate a Vuser script.

1. **Initialize the regeneration**

   Select **Record > Regenerate Script**. VuGen issues a warning indicating that all manual changes will be overwritten.

2. **Modify regenerate options - optional**

   Click **Options** to open the **Regenerate Options** dialog box.

   In a multiple protocol script, you can modify which protocols you want to record when the script is regenerated from the **General > Protocols** node. For user interface details, see "General > Protocol Node" on page 289.

   To change the Script options, select the **General > Script** node and select or clear the appropriate check box. For user interface details, see "General > Script Node" on page 292.

Note that if a Flex, Silverlight, or Java over HTTP Vuser script encounters errors during the code generation phase, VuGen will show the errors in the Error pane. This Error pane displays

details about each error, as well as recommended actions. Follow the recommended actions and regenerate the script.

How to Create or Open a Vuser Script

This task describes various ways to create a new Vuser script or open an existing Vuser script.

Note: Do not name scripts *init*, *run* or *end*, since these names are used by VuGen.

Use the protocol advisor to select a script type

You can use the protocol advisor to produce a list of relevant types of Vuser scripts based on your application. For details, see Protocol Advisor.

Create a new script

Select **File** > **New Script and Solution**, select Single or Multiple protocol from the category list , and then select a protocol from the protocol list. For user interface details, see "Create a New Script Dialog Box" on page 120.

Create or open a script from a template

For task details, see "How to Create and Open Vuser Script Templates" on page 111.

Open a regular script

To open a script stored on your local machine or a network drive, select **File > Open > Script/Solution**.

Open or work with a .zip script

You can unzip or work with a script in .zip format. For task details, see "How to Work with .zip Files" on page 112.

Open a script stored in Application Lifecycle Management

You can store scripts on HP ALM and modify them in VuGen. For more information, see "Working with Application Lifecycle Management" on page 222.

Record Dialog Box

This dialog box enables you to record your business process.

To access	• **VuGen > Record > Record** • [VuGen] **Start Recording** button
Important information	This dialog box is dynamic and changes according to the options you select as well as the protocol you are using.
Relevant tasks	• "How to Record a Vuser Script" on page 111 • "Scripting Options" on page 71

User interface elements are described below:

All Protocols (except Java)

UI Element	Description
	Collapse. Collapses the Start Recording dialog box. The following settings are displayed: • Application type • Recorded application • Record into action • URL address (Web) or Program arguments (Windows) The Start Recording dialog box appears collapsed when you re-record a script.
	Expand. Expands the Start Recording dialog. The Start Recording dialog box appears expanded when you record a new script.
Record into action	The section into which you want to record. You can add your own action. Type the action name in the **Record into action** field and click **Add**. The new action is added to the script.
Application type	The application type used to record your business process. You can record: • **Web.** For example Web and Oracle NCA record web applications. • **Windows**. For example, Windows Socket Vusers records a windows application.
Recorded application	Select the path of the browser application, or windows application to record. The following browsers are supported: • Internet Explorer • Firefox
URL address	The starting URL address. This option is only displayed for internet applications.
Program arguments (Windows applications only)	Specify the command line argument for the executable specified in **Recorded application**. For example, if you specify `plus32.exe` (recorded application) with the command line options `peter@neptune`, it connects the user `Peter` to the server `Neptune` when starting `plus32.exe`.

, continued

Start Recording	You can record your business process either: • **Immediately** - Recording starts as soon as you click the Start Recording button. • **In delayed mode** - In the following instances, it may not be advisable to record immediately: ▪ If you are recording multiple actions, in which case you only need to perform the startup in one action. ▪ In cases where you want to navigate to a specific point in the application before starting to record. ▪ If you are recording into an existing script.
Working directory	For applications that require you to specify a working directory.
Recording Options...	Opens the Recording Options dialog box. For user interface details, "Recording Options" on page 258.
Start Recording button	Instantly begin to record your business process.
Delayed Start button	You can browse or go to the point where you would like to begin recording your business process and then select record.

Java Protocols

UI Element	Description
Recording Options...	Opens the Recording Options dialog box. For user interface details, see "Recording Options" on page 258.
Working directory	A working directory is necessary only if your application must know the location of the working directory (for example, reading property files or writing log files).
Record into action	The section into which you want to record.
URL	The URL to start recording.
Parameters	Any additional parameters that your application requires.
Main Class	This option is only present for Java Application type applications.
Internet Explorer path	This option is only present for Internet Explorer type applications.
Executable\Batch	This option is only present for Executable\Batch type applications.

, continued

Application type	• **Java applet** to record a Java applet through Sun's applet viewer. • **Java application** to record a Java application. • **Internet Explorer** to record an applet within a browser. • **Executable/Batch** to record an applet or application that is launched from within a batch file or the name of an executable file. • **Listener** to instruct VuGen to wait for the batch file that initializes the configuration and runs an application before recording. This mode requires you to define the system variable _JAVA_OPTIONS as --**Xrunjdkhook** using jdk1.2.x and higher. (For JDK 1..x, define the environment variable **_classload_hook=JDKhook**. For JDK 1.6 set _JAVA_OPTIONS as **-agentlib:jdhook**.)

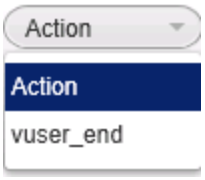
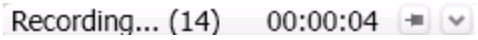
Floating Recording Toolbar

The floating recording toolbar enables you to control the recording of your script and provides easy access to common script commands.

UI example	
To access	The floating recording toolbar automatically displays when script recording begins.
Important information	• The floating recording toolbar is dockable. For details, see "VuGen Layouts - Overview" on page 48. • You can pin the toolbar with the button. • You can expand or collapse the toolbar with the button.
Relevant tasks	"How to Create or Open a Vuser Script" on page 114

User interface elements are described below:

UI Element	Description
	Continue recording the script after recording has been paused.
	Stop recording the script.
	Pause recording.

UI Element	Description
	Select an action to record into.
	Create a new action to record into.
	Insert a **Start Transaction** step into your script. For details, see "Transaction Overview" on page 202.
	Insert a **End Transaction** step into your script. For details, see "Transaction Overview" on page 202.
	Insert a **Rendezvous point** step into your script. For details, see "Rendezvous Points" on page 203.
	Insert comments into your script.
	Insert a **Text Check** step into your script. For details, see "Text and Image Verification" on page 673.
	The title bar includes the following information and commands: • How many events have been recorded into your script. • The time elapsed since recording began excluding time the script was paused. • Click the docking button to pin or unpin the toolbar • Click the Collapse/Expand button to display command icons.

Files Generated During Recording

Assuming that the recorded script has been given the name **vuser** and is stored under **c:\tmp**. The following is a list of the more important files that are generated after recording:

File Name	Details

vuser.usr	Contains information about the virtual user: type, AUT, action files, and so forth.
vuser.bak	A copy of Vuser.usr before the last save operation.
default.cfg	Contains a listing of all run-time settings as defined in the VuGen application (think time, iterations, log, web).
vuser.asc	The original recorded API calls.
vuser.grd	Contains the column headers for grids in database scripts.
default.usp	Contains the script's run logic, including how the actions sections run.
init.c	Exact copy of the Vuser_init function as seen in the VuGen main window.
run.c	Exact copy of the Action function as seen in the VuGen main window.
end.c	Exact copy of the Vuser_end function as seen in the VuGen main window.
vdf.h	A header file of C variable definitions used in the script.
\Data	The Data folder stores all of the recorded data used primarily as a backup. Once the data is in this folder, it is not touched or used. For example, Vuser.c is a copy of run.c.

Example of Vuser.usr File

```
[General]
Type=Oracle_NCA
DefaultCfg=default.cfg
BuildTarget=
ParamRightBrace=>
ParamLeftBrace=<
NewFunctionHeader=0
MajorVersion=5
MinorVersion=0
ParameterFile=nca_test3.prm
GlobalParameterFile=
[Transactions]
Connect=
[Actions]
vuser_init=init.c
Actions=run.c
vuser_end=end.c
```

Example of default.cfg File

```
[General]
XlBridgeTimeout=120
[ThinkTime]
Options=NOTHINK
Factor=1
LimitFlag=0
Limit=1
```

```
[Iterations]
NumOfIterations=1
IterationPace=IterationASAP
StartEvery=60
RandomMin=60
RandomMax=90
[Log]
LogOptions=LogBrief
MsgClassData=0
MsgClassParameters=0
MsgClassFull=0
```

Create a New Script Dialog Box

This dialog box enables you to create a new Vuser script.

To access	Do one of the following: • **VuGen > File > New Script and Solution** • **VuGen > File > Add > New Script** • Click the  button on the VuGen toolbar.
Relevant tasks	• "How to Create or Open a Vuser Script" on page 114 • "How to Record a Vuser Script" on page 111

User interface elements are described below:

UI Element	Description
Category	The category identifies the type of script you would like to create: **Multiple Protocols:** Create a script using multiple protocols. **Single Protocol:** Create a script using the selected protocol. Single Protocol is the default category.
Protocol	The Protocol list displays protocols based on the category you select. **Category** / **Protocol List displays...** **Multiple Protocols** — A list of protocols with a check box to the left of each protocol. Select a check box to include the protocol in the Vuser script. **Single Protocol** — A list of protocols. You can select one protocol.

, continued

Filter	Enables you to filter the Protocol list by entering text. For example, if you type "Java" into the Filter box, the Protocol list will display only those protocols that include the word Java.
Script Name	Enables you to specify the name of your script.
Location	Enables you specify the file location of your script. You can use the browse button to navigate to a location on your file system. **Tools > General > Projects and Solutions** enables you to specify a default location.
Solution Name	This option is displayed only when a solution is not open in the Solution Explorer. You can specify a name for the solution. If you leave it blank, the default name is 'Untitled'.
Create a folder for this solution	Enables you to create a folder for your solution.
Solution Target	Displays the file path of the solution.
Protocol Advisor	Enables you to access the Protocol Advisor dialog box. For details, see "Protocol Advisor Dialog Box" on page 105
	Displays the protocols in list view.
	Displays the protocols in icon view.

Replaying and Debugging Vuser Scripts

Replay Overview

After creating a Vuser script, you replay it in standalone mode, directly from the VuGen interface. This tests the basic functionality, as well as helps you to uncover any errors or issues that need to be addressed, such as dynamic values which need parameterization. When the replay is successful, you are ready to integrate the script into a LoadRunner scenario.

Think Time

The time that a user waits between performing successive actions is known as the *think time*. Vusers use the **lr_think_time** function to emulate user think time. When you record a Vuser script, VuGen records the actual think times and inserts appropriate **lr_think_time** statements into the Vuser script. You can edit the recorded **lr_think_time** statements, and manually add more **lr_think_time** statements to a Vuser script. For task details, see "How to Insert Steps into a Script" on page 207.

Note: When you record a Java Vuser script, **lr_think_time** statements are not generated in the

Vuser script.

You can use the run-time settings to influence how the **lr_think_time** statements operate when you execute a Vuser script. For user interface details, see "General > Think Time Node" on page 361.

Debugging Features

After creating a Vuser script, you can check to see if the script runs properly. Using VuGen's debug panes, you can debug your scripts using VuGen's debugging capabilities.VuGen contains a number of features to help debug your Vuser scripts. You can access most of these script debugging features from the VuGen toolbar.

Note: The debugging features are available for C-based Vuser protocols. The features are not available for VBscript and VB Application type Vuser protocols.

Running a Vuser script

To run a Vuser script until the end of the script or until the next breakpoint, perform one of the following:

- Select **Replay > Run**.
- Click the **Run** button  on the Vuser toolbar.
- Press **F5**.

Note that the status of the Vuser script execution appears in the lower left corner of VuGen. The script execution status may be **Ready**, **Running**, or **Paused**.

- To stop a script that is running, click the **Stop Replay**  button on the VuGen toolbar.
- To pause a script that is running, click the **Pause**  button on the VuGen toolbar.
- To continue running a script that is paused, click the **Continue**  button on the VuGen toolbar.

The Run Step by Step Command

The **Run Step by Step** command runs the script one line at a time. This enables you to follow the script execution. The **Run Step by Step** command starts the script replay, and then and pauses it on the first line of the script, usually in the vuser_init() action.

To run the script step by step, perform one of the following:

- Select **Replay > Run Step by Step**.
- Click the **Run Step by Step** button  on the VuGen toolbar.
- Press **F10**.

Note that the **Run Step by Step** button is available only while a script is being replayed.

Animated Run

You can run a Vuser script in animated mode or non-animated mode. When you run in animated mode, VuGen highlights the line that is running in the Vuser script. When you run in non-animated mode, VuGen executes the Vuser script, but does not indicate the line being executed.

To toggle between the animated mode and the non-animated mode:

- Click the **Animated Run**  button on the VuGen toolbar.
- Click **Replay > Animated Run**.

Note that the **Animated Run** button is available only while a script is being replayed.

For details on how to set the run mode using VuGen options, see "Scripting Options" on page 71.

Breakpoints

Breakpoints pause script execution at specified points in the script. This enables you to examine the effects of the script on your application at pre-determined points during script execution.

- For concept details on breakpoints, see "Working with Breakpoints" on page 125.
- For task details, see "How to Debug Scripts with Breakpoints" on page 130.

Bookmarks

When working in Script view, VuGen lets you place bookmarks at various locations within your script. You can navigate between the bookmarks to help analyze and debug your code.

- For task details, see "How to Use Bookmarks" on page 131.

Watching Variables

The Watch pane enables you to monitor variables and expressions while a script runs. You can monitor variables and expressions only when execution of a Vuser script is in the Paused state. To display the Watch pane, click **View > Debug > Watch**. For details on using the Watch pane, see "Watching Expressions and Variables" on page 126.

Go To Commands

- To navigate around a script using breakpoints, you can use the **Go To Source** command. For details, see "How to Debug Scripts with Breakpoints" on page 130.
- To navigate around a script using bookmarks, you can use the **Next Bookmark** and **Previous Bookmark** commands. For details, see "How to Use Bookmarks" on page 131.

If you want to examine the Replay log messages for a specific step or function, right-click the step in the Editor and select **Go To Step in Replay Log.** VuGen places the cursor at the start of the corresponding step in the Output pane's Replay log.

Error Handling

You can specify how a Vuser handles errors during script execution. By default, when a Vuser detects an error, the Vuser stops executing the script. You can instruct a Vuser to continue with the next iteration when an error occurs using one of the following methods:

- Using run-time settings. You can specify the **Continue on Error** run-time setting. The **Continue on Error** run-time setting applies to the entire Vuser script. You can use the **lr_continue_on_error** function to override the **Continue on Error** run-time setting for a portion of a script.
- Using the **lr_continue_on_error** function. The **lr_continue_on_error** function enables you to control error handling for a specific segment of a Vuser script. To mark the segment, enclose it with `lr_continue_on_error(1);` and `lr_continue_on_error(0);` statements. The new error settings apply to the enclosed Vuser script segment. See the paragraphs below for details.

For example, if you enable the Continue on Error run-time setting and a Vuser encounters an error during replay of the following script segment, the Vuser continues executing the script:

```
    web_link("EBOOKS",
        "Text=EBOOKS",
        "Snapshot=t2.inf",
        LAST);
    web_link("Find Rocket eBooks",
        "Text=Find Rocket eBooks",
        "Snapshot=t3.inf",
        LAST);
```

To instruct the Vuser to continue on error for a specific segment of the script, enclose the segment with the appropriate lr_continue_on_error statements:

```
lr_continue_on_error(1);
    web_link("EBOOKS",
        "Text=EBOOKS",
        "Snapshot=t2.inf",
        LAST);
    web_link("Find Rocket eBooks",
        "Text=Find Rocket eBooks",
        "Snapshot=t3.inf",
        LAST);
lr_continue_on_error(0);
```

Bookmarks Overview

When you edit a Vuser script, you can use bookmarks to navigate between sections of the script. When you add a bookmark, a bookmark icon is added to the left of the selected line in your script.

The Bookmarks pane displays a list of all bookmarks that exist in the Vuser script. Using the Bookmarks pane, you can:

- Navigate to the location of the bookmark in your script.
- Navigate between consecutive bookmarks in the pane.
- Delete an individual bookmark.
- Clear all bookmarks.

For task details, see "How to Use Bookmarks" on page 131.

Additional Debugging Information

General Debugging Tip

VuGen can be used as a regular text editor. You can open any text file in it and edit it. When an error message is displayed during replay in the output window below, you can double click on it and VuGen jumps the cursor to the line of the test that caused the problem. You can also place the cursor on the error code and press F1 to view the online help explanation for the error code.

Using C Functions for Tracing

You can use the C interpreter trace option (in version 230 or higher) to debug your Vuser scripts. The `ci_set_debug` statement allows trace and debug to be turned on and off at specific points in the script.

```
ci_set_debug(ci_this_context, int debug, int trace);
```

For example, you could add the following statements to your script:

```
ci_set_debug(ci_this_context, 1, 1) /* turn ON trace =; debug */
ci_set_debug(ci_this_context, 0, 0) /* turn OFF trace =; debug */
```

Additional C Language Keywords

When you run a C script in VuGen, its parser uses the built-in C interpreter to parse the functions in the script. You can add keywords that are not part of the standard parser's library. By default, several common C++ keywords are added during installation, such as *size_t* and *DWORD*. You can edit the list and add additional keywords for your environment.

Add Additional Keywords

1. Open the vugen_extra_keywords.ini file, located in your machine's <Windows> or <Windows>/System directory.
2. In the `EXTRA_KEYWORDS_C` section, add the desired keywords for the C interpreter.

   The file has the following format:

   ```
   [EXTRA_KEYWORDS_C]
   FILE=
   size_t=
   WORD=
   DWORD=
   LPCSTR=
   ```

Examining Replay Output

Look at the replay output (either from within VuGen, or the file output.txt representing the output of the VuGen driver). You may also change the run-time settings options in VuGen to select more extensive logging in order to obtain a more detailed log output of the replayed test.

Working with Breakpoints

VuGen lets you include breakpoints in you Vuser scripts to help you to debug the scripts. Breakpoints pause script execution at specified points in the script. This enables you to analyze the

effects of the script on your application at pre-determined points during script execution. For task details, see "How to Debug Scripts with Breakpoints" on page 130. A breakpoint symbol ( ) in the left margin of the script indicates the presence of a breakpoint. In addition, VuGen highlights the line in the script.

You can disable a breakpoint if the breakpoint is temporarily not required. A white dot inside the Breakpoint symbol indicates a disabled breakpoint ( ). When a breakpoint is disabled, script execution continues at the disabled breakpoint and is paused at the following enabled breakpoint. You use the Breakpoints pane to enable and disable breakpoints. In addition, the breakpoints pane enables you to delete an existing breakpoint or delete all existing breakpoints. To display the Breakpoints pane, click **View > Debug > Breakpoints**.

To run a script with breakpoints, begin running the script as usual. VuGen pauses script execution when it reaches a breakpoint. You can examine the effects of the script run up to the breakpoint, make any necessary changes, and then restart the script from the breakpoint.

To resume execution, select **Replay > Run**. Once restarted, the script continues until it encounters another breakpoint or the end of the script.

Watching Expressions and Variables

VuGen's Watch pane enables you to monitor variables while a script runs. The list of variables that you want to watch is known as the watch list, and is displayed in the watch pane. To display the Watch pane, click **View > Debug > Watch**. You can add only variables to the watch list - you cannot add expressions to the watch list. You can add, edit, or remove variables within the watch list by using the Watch pane's toolbar buttons. You can sort the columns in the watch pane by expression, value, or type name by clicking the column headers. For details on other debugging features in VuGen, see "Debugging Features" on page 122.

Note: You can monitor variables only when execution of a Vuser script is in the **Paused** state.

Adding a New Watch to the Watch List

You can add a new watch expression only when execution of a Vuser script is in the **Paused** state.

To add a new watch:

1. Click **View > Debug > Watch** to open the Watch pane.
2. Click the **Add Watch** button  . The Add New Watch dialog box opens.
3. In the **Expression** field, enter the variable that you want to watch, and then click **OK**. VuGen adds the variable to the list of expressions in the watch list.

Note: You can add only variables to the watch list - you cannot add expressions to the watch list.

Editing a Watch Expression

Note: You can edit a watch expression only when execution of a Vuser script is in the **Paused** state.

To edit a watch expression:

1. Click **View > Debug > Watch** to open the Watch pane.
2. In the watch list, select the expression that you want to edit, and then click the **Edit Watch Expression** button  . The Edit Watch dialog box opens.
3. In the **Expression** field, modify the existing variable as required, and then click **OK**. VuGen displays the modified variable in the list of variables in the watch list.

Deleting a Watch Expression

Note: You can delete a watch expression only when execution of a Vuser script is in the **Paused** state.

To delete a watch expression:

1. Click **View > Debug > Watch** to open the Watch pane.
2. In the Watch pane, select the expression that you want to delete, and then click the **Delete Watch** button  . VuGen deletes the selected expression from the list of expressions in the watch list.

Deleting All Watch Expressions From the Watch List

Note: You can delete watch expressions only when execution of a Vuser script is in the **Paused** state.

To delete all watch expressions from the watch list:

1. Click **View > Debug > Watch** to open the Watch pane.
2. Click the **Delete All Watches** button  . VuGen deletes all the expressions from the watch list.

Debugging Web Vuser Scripts

VuGen provides an additional tool to help you debug Web Vuser scripts—the Results Summary report.

You can specify whether or not a Web Vuser generates a Results Summary report during script execution. The Results Summary report summarizes the success or failure of each step in the Web Vuser scripts and allows you to view the Web page returned by each step. For additional details on working with the Results Summary report, select **Replay > Test Results** and click F1 to open the online help.

For more user interface information, see "Scripting Options" on page 71.

Note: Transaction times may be increased when a Vuser generates a Results Summary report. Vusers can generate Results Summary reports only when run from VuGen. When you run a script from the Controller or Business Process Monitor, Vusers do not generate reports.

How to Replay a Vuser Script

This task describes how to replay a Vuser script.

1. **Configure the run-time settings and replay options**

   Specify the desired options in Scripting pane of the Options dialog box. For user interface details, see Scripting Options.

2. **Replay the script**

   Select **Replay** > **Run** to replay the script.

3. **View the logs for detailed information**

   You can view detailed information about how your script behaved during the recording and replay stages in the output window. For user interface details, see Output Window.

How to Run a Vuser Script from a Command Prompt

This task describes how to test a Vuser script from a command prompt or from the Windows Run dialog box—without the VuGen user interface

Run a Script From a Command Line or the Run Dialog Box

1. Select **Start** > **Programs > Command Prompt** to open a **Command Prompt** window, or select **Start** > **Run** to open the Run dialog box.
2. Type the following and press **Enter**:

   ```
   <installation_dir>/bin/mdrv.exe -usr <script_name> -vugen_win 0
   ```

   script_name is the full path to the *.usr* script file, for example, **c:\temp\mytest\mytest.usr.**

   The **mdrv** program runs a single instance of the script without the user interface. Check the output files for run-time information.

3. You can specify arguments to pass to your script by using the following format:

   ```
   script_name   -argument argument_value   -argument argument_value
   ```

4. You can specify the load generator, as well as indicate the number of times to run the script as indicated by the following example:

   ```
   script1  -host pc4  -loop 5
   ```

   For more information on the command line parsing functions, or for details on including arguments on a command line, see the Function Reference (**Help > Function Reference**).

How to Run a Vuser Script from a Linux command line

When using VuGen to develop Linux-based Vusers, you must check that the recorded script runs on the Linux platform. This task describes how to perform this check and run a Vuser script from a Linux command.

1. **Verify that the script replays in VuGen**

   Replay the script in VuGen to verify that the script works in windows before attempting to run it in Linux. This is recommended because it is easier to edit and debug the script in VuGen. For task details, see "How to Replay a Vuser Script" on previous page.

2. ### Copy the script files to the Linux server

   Transfer the script files to the Linux server

3. ### Check the Vuser setup on the Linux machine by using verify_ generator.

   If you intend to run all of the Vusers on one host, type:

   ```
   verify_generator
   ```

   The verify_generator either returns **OK** when the setting is correct, or **Failed** and a suggestion on how to correct the setup.

   For detailed information about the verify checks type:

   ```
   verify_generator [-v]
   ```

   The verify utility checks the local host for its communication parameters and its compatibility with all types of Vusers. It checks the following items in the Vuser environment:

   - at least 128 file descriptors
   - proper **.rhost** permissions: **-rw-r--r--**
   - the host can be contacted using rsh to the host. If not, checks for the host name in **.rhosts**
   - **M_LROOT** is defined
   - .**cshrc** defines the correct **M_LROOT**
   - .**cshrc** exists in the home directory
   - the current user is the owner of the **.cshrc**
   - a LoadRunner installation exists in **$M_LROOT**
   - the executables have executable permissions
   - **PATH** contains **$M_LROOT/bin**, and **/usr/bin**
   - the **rstatd** daemon exists and is running

4. ### Run the script

   Run the script in standalone mode from the Vuser script folder, using the **run_db_vuser** shell script:

   ```
   run_db_vuser.sh <commands> script_name.usr
   ```

   The *run_db_vuser* shell script has the following command line options:

Command	Description
--help	Display the available options. (This option must be preceded by two dashes.)
-cpp_only	Run cpp only (pre-processing) on the script.
-cci_only	Run cci only (pre-compiling) on the script to create a file with a .ci extension. You can run cci only after a successful cpp.

-driver ***driver_path***	Use a specific driver program. Each database has its own driver program located in the /bin folder. For example, the driver for **CtLib** located in the /bin folder, is **mdrv**. This option lets you specify an external driver.
-exec_only	Execute the Vuser .ci file. This option is available only when a valid .ci file exists.
-ci ***ci_file_name***	Execute a specific .ci file.
-out ***output_path***	Place the results in a specific folder.

By default, *run_db_vuser.sh* runs **cpp**, **cci**, and **execute** in verbose mode. It uses the driver in the VuGen installation\bin folder, and saves the results to an output file in the Vuser script folder. You must always specify a *.usr* file. If you are not in the script folder, specify the full path of the *.usr* file.

For example, the following command line executes a Vuser script called test1, and places the output file in a folder called results1. The results folder must be an existing folder—it will not be created automatically:

```
run_db_vuser.sh-out /u/joe/results1  test1.usr
```

How to Debug Scripts with Breakpoints

The following steps describe how to work with breakpoints. For concept details, see "Working with Breakpoints" on page 125.

Add a Breakpoint

To add a breakpoint:

Locate the cursor in the script where you want to insert the breakpoint and then do one of the following:

- Select **Replay > Toggle Breakpoint**.
- Press **F9**.
- Click in the left margin if the script, adjacent to where you want to insert the breakpoint.

The **Breakpoint** symbol () appears in the left margin of the script, and VuGen highlights the line in the script.

Delete a Breakpoint

To delete a breakpoint:

Locate the cursor in the script where you want to delete the breakpoint and then do one of the following:

- Select **Replay > Toggle Breakpoint**.
- Press **F9**.

- Click the breakpoint symbol left margin if the script.

The **Breakpoint** symbol ( ) is removed from the left margin of the script.

Enable/Disable a Breakpoint

To disable a breakpoint:

Click **View > Debug > Breakpoints** to display the Breakpoints pane.

- Select the appropriate **Enable** check box to enable a breakpoint. The **Breakpoint** symbol ( ) appears in the left margin of the script.
- Clear the appropriate **Enable** check box to disable a breakpoint. The **Disabled Breakpoint** symbol ( ) appears in the left margin of the script.

When a breakpoint is disabled, script execution continues at the disabled breakpoint and is paused at the following enabled breakpoint.

Manage Breakpoints

The Breakpoints pane allows you to remove, enable, and disable breakpoints in a Vuser script. For user interface details, see "Breakpoints Pane" on page 96.

Navigate to a specific breakpoint in a Vuser script

To navigate to a specific breakpoint in a Vuser script, perform one of the following:

- In the Breakpoints pane, select the specific breakpoint to which you want to navigate, and then click the **Go to source** button .
- In the Breakpoints pane, double-click the breakpoint to which you want to navigate.

The cursor flashes in the Editor at the start of the line containing the breakpoint.

Run a Script With Breakpoints

Begin running the script as usual. VuGen pauses script execution when it reaches a breakpoint. You can examine the effects of the script run up to the breakpoint, make any necessary changes, and then restart the script from the breakpoint.

To resume execution, select **Replay > Run**. Once restarted, the script continues until it encounters another breakpoint or the end of the script.

How to Use Bookmarks

When working in the Editor, VuGen lets you place bookmarks at various locations within your script. You can navigate between the bookmarks to analyze and debug your code. The following steps describe how to work with bookmarks. Most of the bookmark functionality is available from VuGen's Bookmarks pane. To access the Bookmarks pane, click **View > Bookmarks**.

Create a Bookmark

In the Editor, place the cursor at the desired location and press Ctrl + F2. VuGen places a bookmark icon in the left margin of the script.

Remove a Bookmark

To remove a bookmark, perform one of the following:

- In the Editor, click in the line that contains the bookmark and press Ctrl + F2.
- In the Bookmark pane, select the bookmark that you want to delete and click the **Delete Bookmark** button  .

VuGen removes the bookmark icon from the left margin.

Navigate between Bookmarks

Click **View > Bookmarks** to display the Bookmarks pane.

- To move to the next bookmark, click the **Next Bookmark** button  or press **F2**.
- To return to the previous bookmark, click the **Previous Bookmark** button  or press **Shift + F2**.

You can navigate between bookmarks in the current action only. To navigate to a bookmark in another action, select that action in the left pane and then press F2.

Navigate to a Specific Bookmark in a Vuser Script

In the Bookmarks pane, double-click the specific bookmark to which you want to navigate. The cursor flashes in the Editor at the start of the line containing the bookmark.

Overview of Files Generated During Replay

This section describes what occurs when the Vuser is replayed.

1. The **options.txt** file is created which includes command line parameters to the preprocessor.
2. The file **Vuser.c** is created which contains 'includes' to all the relevant .c and .h files.
3. The c preprocessor **cpp.exe** is invoked in order to 'fill in' any macro definitions, precompiler directives, and so on, from the development files.

   The following command line is used:

   ```
   cpp -foptions.txt
   ```

4. The file **pre_cci.c** is created which is also a C file (**pre_cci.c** is defined in the **options.txt** file). The file **logfile.log** (also defined in **options.txt**) is created containing any output of this process. This file should be empty if there are no problems with the preprocessing stage. If the file is not empty then its almost certain that the next stage of compilation will fail due to a fatal error.
5. The **cci.exe** C compiler is now invoked to create a platform-dependent pseudo-binary file (.ci) to be used by the virtual user driver program that will interpret it at run-time. The cci takes the **pre_cci.c** file as input.
6. The file **pre_cci.ci** is created as follows:

   cci -errout c:\tmp\Vuser\logfile.log -c pre_cci.

7. The file **logfile.log** is the log file containing output of the compilation.
8. The file **pre_cci.ci** is now renamed to **Vuser.ci**.

Since the compilation can contain both warnings and errors, and since the driver does not know the results of this process, the driver first checks if there are entries in the **logfile.log** file. If there are, it then checks if the file **Vuser.ci** has been built. If the file size is not zero, it means that the cci has succeeded to compile - if not then compilation has failed and an error message will be given.

9. The relevant driver is now run taking both the **.usr** file and the **Vuser.ci** file as input. For example:

```
mdrv.exe  -usr c:\tmp\Vuser\.usr    -out c:\tmp\Vuser    -file
c:\tmp\Vuser\Vuser.ci
```

The **.usr** file is needed since it tells the driver program which database is being used. From here it can then know which libraries need to be loaded for the run.

10. The **output.txt** file is created (in the path defined by the 'out' variable) containing all the output messages of the run. This is the same output as seen in both the VuGen runtime output window and the VuGen main lower window.

Example of options.txt file

```
-DCCI
-D_IDA_XL
-DWINNT
-Ic:\tmp\Vuser                  (name and location of Vuser include
files)
-IE:\LRUN45B2\include           (name and location of include files)
-ec:\tmp\Vuser\logfile.log      (name and location of output logfile)
 c:\tmp\Vuser\VUSER.c           (name and location of file to be
processed)
```

Example of Vuser.c file

```
#include "E:\LRUN45B2\include\lrun.h"
#include "c:\tmp\web\init.c"
#include "c:\tmp\web\run.c"
#include "c:\tmp\web\end.c"
```

Correlation/Async Studio

Correlation Studio

Correlation Overview

Correlation is used when a recorded script includes a dynamic value (such as a session ID) and therefore cannot be successfully replayed. To resolve this, you convert the dynamic value into a variable—thereby enabling your script to replay successfully.

For example, many applications and Web sites use the current date and time to identify a session. If you try to replay a script that was recorded on such a site, the script may fail because the current time is different from the recorded time. Correlating the data enables you to save the dynamic data and use it throughout the scenario run.

When a correlation is created, VuGen adds a function that extracts the dynamic value to a parameter. Appropriate occurrences of the original value are replaced with a parameter.

Correlation Tab [Design Studio] Overview

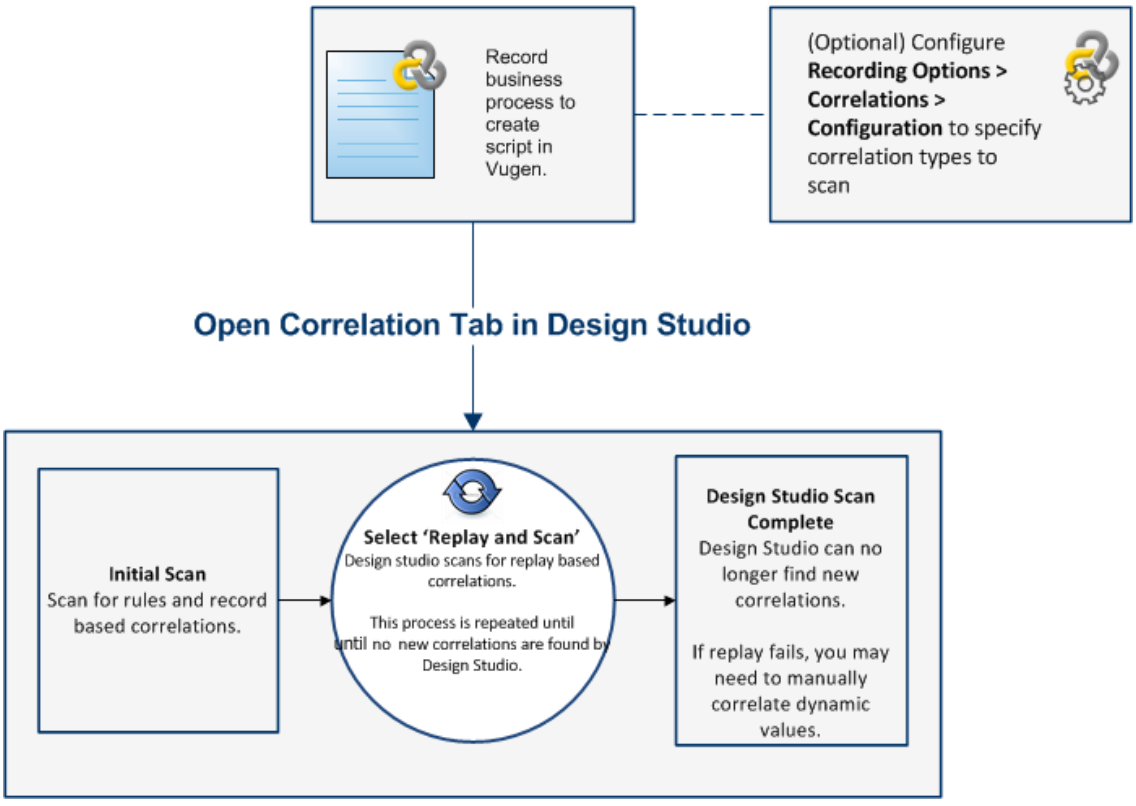
The Correlation tab enables you to correlate and manage dynamic values in your web-based Vuser scripts. To learn more about correlation concepts, see "Correlation Overview" on previous page.

With the **Correlation tab** you can:

- Scan for correlations using rules, record based, and replay based engines
- Correlate both raw and formatted data
- Add and edit rules
- Undo correlations
- Review details pertaining to a specific dynamic value in a snapshot

When you record a script using a web-based protocol, many of the values change dynamically each time a request is sent to the server. An example of a dynamic value is a `sessionID` which may include a date and time stamp of when the web session was opened.

The following flow chart illustrates the process for correlating values in your script using the Correlation tab:



As you can see from the flow chart, the Correlation tab scans for dynamic values using different processes.

Correlation Types

Design Studio uses three processes to automatically find dynamic values that may need to be correlated.

- Rules

  Design Studio first scans for dynamic values that are defined by rules, if the rules scan has been enabled. To learn more, see "Correlation Rules" below.

- Record

  Design Studio scans for dynamic values after code generation. This method can find a significant percentage of dynamic values in your script.

- Replay

  Design Studio scans for dynamic values after replay. This method may need to be repeated several times.

You can select which scan types the Correlation tab should use by configuring **Recording Option > Correlations > Configuration**. In general, it is recommended to enable all scan types.

The following table explains the expected behavior at various script states:

Script State when opening the Correlation tab	Behavior in the Correlation tab (All scan types enabled)
Script contains recorded data.	When Design Studio is opened, it will scan for rule and record based correlations. You can then replay and scan for replay based correlations. Repeat this process until the Design Studio no longer finds new correlations.
Script contains recorded data and has been replayed.	When the Correlation tab is opened, it will scan for all correlation types. You can then replay and scan for additional replay based correlations. Repeat this process until Design Studio no longer finds new correlations.

Correlation Rules

If you know the dynamic values that need to be correlated before recording, you can create correlation rules that will automatically identify those values while you record. If "**Automatically apply correlation rules**" is selected in the **Recording Options > Correlations > Configuration** node, values found based on rules will automatically be correlated. Additionally, there are some correlation rules that come pre-defined in VuGen for supported application servers. You can enable or disable rules in "Correlations > Rules" on page 276.

Snapshot Details and Occurrences

Design Studio provides details on each snapshot step that contains dynamic values. These details can help you determine which values to correlate in your script. In addition to the snapshot details, the Correlation tab, displays all occurrences of the dynamic value in your script. You can select specific occurrences to correlate or correlate all. For details, see "Design Studio [Correlation Tab] Dialog Box" on page 168.

Determining Which Values to Correlate

Once you generate a list of differences, you need to determine which ones to correlate. If you mistakenly correlate a difference that did not require correlation, your replay may be adversely affected.

The following strings most probably require correlation:

- **Login string.** A login string with dynamic data such as a session ID or a timestamp.
- **Date/Time Stamp.** Any string using a date or time stamp, or other user credentials.
- **Common Prefix.** A common prefix, such as **SessionID** or **CustomerID**, followed by a string of characters.

If you are in doubt whether a difference should be correlated, correlate only that difference and then run your script. Check the Replay log to see if the issue was resolved.

You should also correlate differences in which some of the recorded and replayed strings are identical, but others differ. For example, SessionID strings with identical prefixes and suffixes, but different characters in between, should be correlated.

Modifying Saved Parameters

After you save a value to a parameter, you may need to modify it before using it in your script. If you need to perform arithmetical operations on a parameter, you must change it from a string to an integer using the **atoi** or **atol** C functions. After you modify the value as an integer, you must convert it back to a string to use the new variable in your script.

In the following WinSock example, the data at offset 67 was saved to the parameter, **param1**. Using **atol**, VuGen converted the string to a long integer. After increasing the value of **param1** by one, VuGen converted it back to a string using sprintf and saved it as a new string, **new_param1**. The value of the parameter is displayed using **lr_output_message**. This new value may be used at a later point in the script.

```
lrs_receive("socket2", "buf47", LrsLastArg);lrs_save_param("socket2",
                    NULL, "param1", 67, 5);
lr_output_message ("param1: %s", lr_eval_string("<param1>"));
sprintf(new_param1, "value=%ld", atol(lr_eval_string("<param1>")) +
1);
lr_output_message("ID Number:"%s" lr_eval_string("new_param1"));
```

Correlation vs. Parameterization

Parameterization is used when you want to take a value and turn it into a variable in order to make your script more realistic. For example, if you are filling out a form on a website, you may want to vary the value entered for a particular field.

Correlation is used when a recorded script includes a dynamic value (such as a session ID) and cannot replay. To resolve this, you make the dynamic value into a variable thereby enabling your script to replay successfully.

Wdiff Correlation Utility

The Wdiff Utility lets you compare recorded Vuser scripts and replay results to determine which values need to be correlated.

To use *WDiff* effectively, you record the identical operation twice, and compare the scripts (or data files for Tuxedo, WinSock, and Jolt). WDiff displays differences in yellow. Note that not all differences indicate a value to correlate. For example, certain receive buffers that indicate the time of execution do not require correlation.

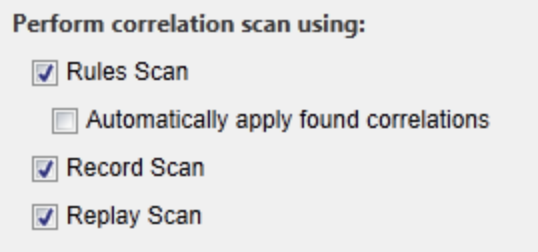
For task details, see "How to Search for Values that Need Correlation" on page 161.

How to Correlate Scripts Using Design Studio

This topic describes how to use the Correlation tab to correlate Vuser scripts.

Prerequisites

1. Record a script using one of the following protocols:
   - Web HTTP/HTML
   - Flex
   - RTMP/RTMPT
   - Citrix
   - SAP Web
2. Verify that **Record > Recording Options > Correlations > Configuration** has all scan types enabled.



Using the Correlation tab

1. **Initial Scan:** Click the **Design Studio** button. This will open the Design Studio dialog box which will scan for response (or record based) correlations and apply correlation rules. The progress bar in the dialog box indicates if the initial scan was successful.

   For details on the Correlation tab, see "Correlation Tab [Design Studio] Overview" on page 134.

2. You can select which values you would like to correlate by highlighting the value in the grid and clicking the **Correlate** button.

   When a value is correlated, VuGen adds a **web_reg_save_param_*** function, and saves the original value in a comment in the script.

You can examine the details of the correlation by expanding the Details Chevron in the dialog box. For details, see "Design Studio [Correlation Tab] Dialog Box" on page 168.

For details on the Correlation tab, see "Correlation Tab [Design Studio] Overview" on page 134.

3. You can click the **Add as Rule** button to add a rule type . In addition, you can click the **Edit rule** button to view and edit the corresponding rule if the dynamic value was correlated by a rule. For details, see Recording Options > Correlation > Rules.
4. **Replay and Scan:** After you have correlated the values, click the **Replay and Scan** button. The Correlation tab progress bar indicates if additional correlations have been found. Again, you can select which values you would like to correlate by highlighting the value in the grid and clicking the **Correlate** button. You may need to repeat this step several times.
5. **Design Studio Scan Complete:** When Design Studio no longer finds new correlations, the progress bar will display "Design Studio Scan Complete".

> **Tip:** If Design Studio does not resolve all correlation-based errors, try to resolve them using manual correlation. For details, see "How To Manually Correlate Scripts" below.

How To Manually Correlate Scripts

If the scan for correlation did not resolve all correlation-based errors in your script, you can attempt to manually correlate your script as follows:

1. **Search for values that need correlation manually.** There are a number of ways to manually search for values that need correlation. For details, see "How to Search for Values that Need Correlation" on page 161.
2. **Correlate the value.**

Select one of the following methods:

- **Correlate from snapshots.** Highlight the value to correlate, right-click, and select **Create Correlation**.

  When a value is correlated, VuGen adds the correlation parameter and saves the original value in a comment in the script.

```
252 /* Correlation comment - Do not change!  Original value='1' Name ='CorrelationParameter' Type ='Manual' */
253     lrc_save_rs_param(_Recordset_45,
254         1,
255         2,
256         0,
257         "CorrelationParameter1");
```

- **Manually add correlation functions.** Manually insert the relevant correlation functions into your script. For details, see "How to Correlate Scripts - Web (Manually)" on page 142.

How to Correlate Scripts From a Snapshot

The following steps describe how to correlate scripts from a snapshot.

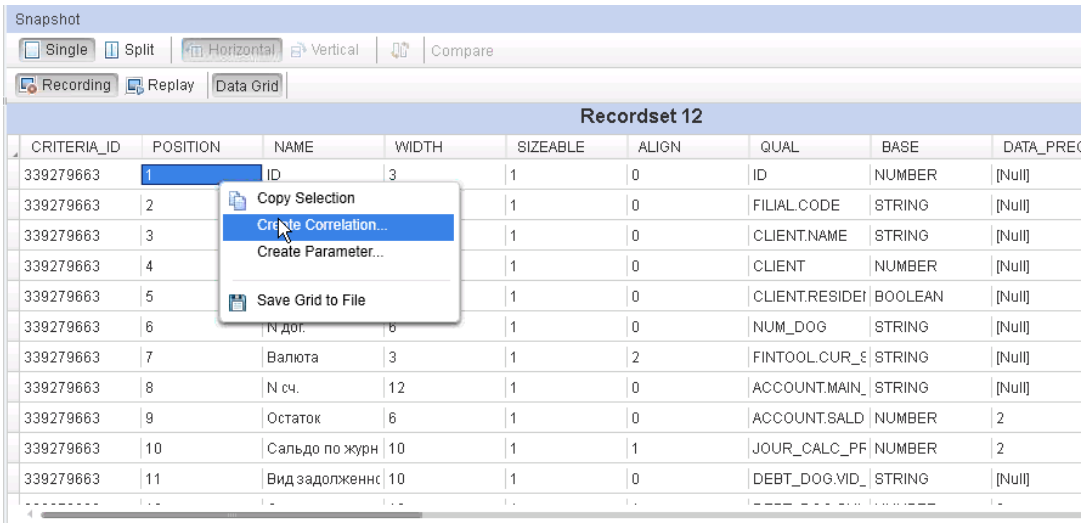
This task applies to the following protocols:

- Database Protocols
- RTMP
- COM Protocols

1. Open the **Output Pane**

   Select **View** > **Output** to display the output tabs at the bottom of the window. Check for errors in the Replay tab. Often, these errors can be corrected by correlation.

2. Select the relevant step in the **Step Navigator**, and view the step in the **Snaphot pane**. Right click the value in the snapshot and select **Create correlation**. This will open the **Design Studio** window.



3. You can select the value you would like to correlate by highlighting it in the grid and clicking the **Correlate** button.

4. When a value is correlated, VuGen adds the correlation parameter, and saves the original value in a comment in the script.

```
/* Correlation comment - Do not change!  Original value='1' Name ='CorrelationParameter' Type ='Manual' */
    lrc_save_rs_param(_Recordset_45,
        1,
        2,
        0,
        "CorrelationParameter");
```

Correlating Winsock Scripts

VuGen's Design Studio provides a user interface for correlating Vuser scripts. Correlation is required when working with dynamic data. A common issue with Winsock Vuser scripts is dynamic ports—ports whose numbers are assigned dynamically. While certain applications always use the same port, others use the next available port. If you try to replay a script and the recorded port is no longer available, your script will fail to replay. To overcome this issue, you must perform correlation—save the actual run-time values and use them within the script.

VuGen uses **lrs_save_param** and **lrs_save_searched_string** functions correlate Winsock scripts. This means that it stores the data that is received for use in a later point within the script. Since correlation stores the received data, it applies only to Receive buffers and not to Send buffers. The

recommended procedure is to select a string of dynamic data within the Receive buffer that you want to correlate. Use that same parameter in a subsequent Send buffer.

Correlating a Winsock script

You use the Snapshot pane to begin correlating Winsock Vuser scripts. Both the Text and the Hex tabs in the Snapshot pane have the correlating functionality.

1. In the Snapshot pane, select the data that you want to correlate.
2. Right-click in the selection, and select **Create Correlation** or **Create Boundary Correlation**. The Design Studio opens and displays the Correlation tab.

   Note that you can click the number of occurrences in the **Replace/Found** column to enable you to choose the exact occurrences that you want to correlate.
3. Click the **Details** bar to display details about the correlation.
4. Make sure that the **Original Snapshot Step** tab is visible. Notice that the type is either **Data Range** or **Boundary Based**.
5. Click **Correlate** to perform the correlation of the Vuser script.
6. Click **Close** to close the Design Studio. Notice that VuGen has inserted the appropriate correlation functions and comments into the script.

For further details on how to use the Design Studio, see "Correlation Tab [Design Studio] Overview" on page 134.

Parameterization vs Correlation

This type of correlation should not be confused with simple parameterization. Simple parameterization (**Design > Parameters > Create New Parameter**) applies only to data within Send buffers. You set up a parameter and assign it several values. VuGen uses the different values in each of the script runs or iterations. For further details, see "Correlation vs. Parameterization" on page 136.

For details on how to manually correlate a Winsock Vuser script, see "How to Correlate Scripts - Winsock (Manually)" below.

How to Correlate Scripts - Winsock (Manually)

This topic describes how to use the Editor to manually correlate a Winsock Vuser script.

1. Insert the **lrs_save_param_ex** statement into your script at the point where you want to save the buffer contents. You can save user, static, or received type buffers.

   ```
   lrs_save_param_ex (socket, type, buffer, offset, length, encoding,
   parameter);
   ```
2. View the buffer contents by selecting **data.ws** in the Action Pane of the main VuGen window (displayed by default in the Editor). Locate the data that you want to replace with the contents of the saved buffer. Replace all instances of the value with the parameter name in parameter braces. The default parameter braces are brackets (< > or ()). You can modify the parameter braces in the **Tools > Options > Scripting > Parameters** tab.

In the following example, a user performed a telnet session. The user used a ps command to determine the process ID (PID), and killed an application based on that PID.

```
frodo:/u/jay>ps
   PID TTY      TIME CMD
14602 pts/18    0:00 clock
14569 pts/18    0:03 tcsh
frodo:/u/jay>kill 14602
[3]    Exit 1                clock
frodo:/u/jay>
```

During execution, the PID of the procedure is different (Linux assigns unique PIDs for every execution), so killing the recorded PID will be ineffective. To overcome this problem, use **lrs_save_param_ex** to save the current PID to a parameter. Replace the constant with the parameter.

3. In the **data.ws** file, determine the buffer in which the data was received, buf47.

```
recv buf47 98
    "\r"
    "\x00"
    "\r\n"
    "   PID TTY      TIME CMD\r\n"
    " 14602 pts/18   0:00 clock\r\n"
    " 14569 pts/18   0:02 tcsh\r\n"
    "frodo:/u/jay>"
.
.
.
send buf58
    "kill 14602"
```

4. In the Actions section, determine the socket used by buf47. In this example it is socket1.

```
lrs_receive("socket1", "buf47", LrsLastArg);
```

5. Determine the offset and length of the data string to save. The offset of the **PID** is 11 and its length is 5 bytes. For additional information about displaying the data, see "Data Buffers" on page 799.

6. Insert an **lrs_save_param_ex** function in the Action section, after the **lrs_receive** for the relevant buffer. In this instance, the buffer is **buf47**. The PID is saved to a parameter called param1. Print the parameter to the output using **lr_output_message**.

```
lrs_receive("socket1", "buf79", LrsLastArg);
lrs_save_param("socket1", "user", buf47, 11, 5, ascii, param1);

lr_output_message ("param1: %s", lr_eval_string("<param1>"));
lr_think_time(10);
lrs_send("socket1", "buf80", LrsLastArg);
```

7. In the data file, data.ws, determine the data that needs to be replaced with a parameter, the PID.

```
send buf58
    "kill 14602"
```

8. Replace the value with the parameter, enclosed in angle brackets.

```
send buf58
    "kill <param1>"
```

How to Correlate Scripts - Web (Manually)

This task describes how to correlate web scripts manually by modifying the code.

1. ### Locate the string and its details

   Identify the statement that contains dynamic data and the patterns that characterize the locations of the data. These patterns may be boundaries or xpaths.

   a. **Identify Patterns using Boundaries**

   Use these guidelines to determine and set the boundaries of the dynamic data:

   - Analyze the location of the dynamic data within the HTTP response.
   - Identify the string that is immediately to the left of the dynamic data. This string defines the left boundary of the dynamic data.
   - Identify the string that is immediately to the right of the dynamic data. This string defines the right boundary of the dynamic data.
   - The right and left boundaries should be as unique as possible to better locate the strings.
   - **web_reg_save_param_ex** looks for the characters between (but not including) the specified boundaries and saves the information beginning one byte after the left boundary and ending one byte before the right boundary. **web_reg_save_param_ex** does not support embedded boundary characters.
     For example, if the input buffer is `{a{b{c}` and "**{**" is specified as a left boundary, and "**}** " as a right boundary, the first instance is c and there are no further instances—it found the right and left boundaries but it does not allow embedded boundaries, so "c" is the only valid match.

   By default, the maximum length of any boundary string is 256 characters. Include a **web_set_max_html_param_len** function in your script to increase the maximum permitted length. For example, the following function increases the maximum length to 1024 characters:

   These length restrictions do not apply where either the left or right boundaries are blank.

   b. **Identify Patterns using Xpaths**

   Use the snapshot pane to manually search for the xpath of the desired string.

   By default, the maximum length of any boundary string is 256 characters Include a **web_set_max_html_param_len** function in your script to increase the maximum permitted length. For example, the following function increases the maximum length to 1024 characters:

   These length restrictions do not apply where either the left or right boundaries are blank.

2. ### Add web_reg_save_param_* function

   Add a **web_reg_save_param_ex or web_reg_save_param_xpath** function into the script before the statement that contains the dynamic data.

   a. **web_reg_save_param_ex**

   This function searches server responses in web steps for the left boundary following by the string and the right boundary and saves the string to a parameter named in the function's argument. After finding the specified number of occurrences, **web_reg_save_param_ex** does not search any more responses. For more information, see the Function Reference **(Help > Function Reference**).

   b. **web_reg_save_param_xpath**

   This function searches server responses in web steps for a specified xpath. The string located in specified xpath is saved to a parameter named in the function's argument. For more information, see the Function Reference (**Help > Function Reference**).

3. ### Replace data with parameter

   Select **Edit** > **Replace** from the VuGen main window to display the Search and Replace dialog box. Search the entire script for the dynamic data and replace it with a parameter. Give the parameter any name and enclose it with braces: `{param_name}`. You can include a maximum of 64 parameters per script.

How to Correlate Scripts - Tuxedo Protocol

The following steps describe how to correlate Tuxedo Vuser scripts.

To correlate statements, you modify your recorded script within the VuGen editor using one of the following LRT functions:

- **lrt_save[32]_fld_val.** Saves the current value of an FML or FML32 buffer (a string in the form "name=<NAME>" or "id=<ID>") to a parameter.
- **lrt_save_parm.** Saves a portion of a character array (such as a STRING or CARRAY buffer) to a parameter.
- **lrt_save_searched_string.** Searches for an occurrence of a string in a buffer and saves a portion of the buffer, relative to the string occurrence, to a parameter.

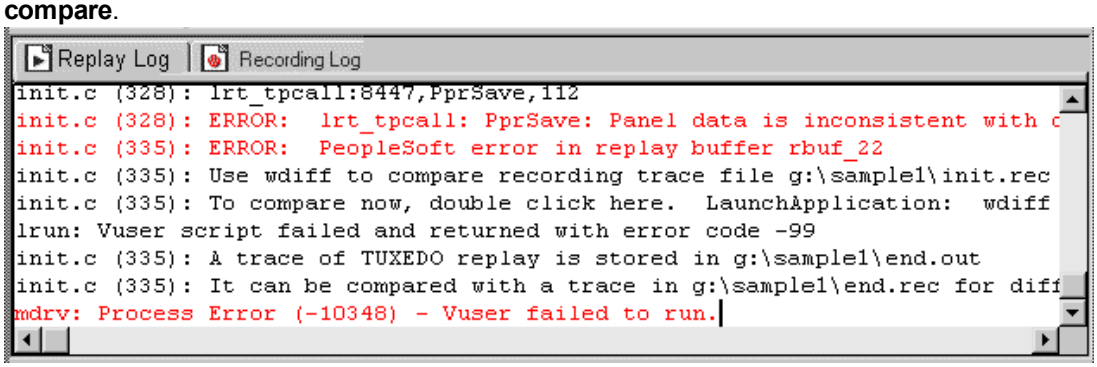
For additional information about the syntax of these functions, see the Function Reference (**Help > Function Reference**).

Determine Values That Need Correlation

When working with CARRAY buffers, VuGen generates log files during recording (with the **.rec** extension) and during replay (with the **.out** extension) which you can compare using the Wdiff utility. You can look at the differences between the recording and replay logs to determine which portions of CARRAY buffers require correlation. The following steps describe how to compare the log files.

1. Select **View** > **Output** to display the execution log and recording log for your script.
2. Examine the Replay Log tab.

The error message should be followed by a statement beginning with the phrase: **Use wdiff to compare**.



3. Double-click on the statement in the execution log to start the **Wdiff** utility.

   For more information, see Wdiff Utility.

Correlate for FML/FML32 Buffers

Use **lrt_save_fld_val** or **lrt_save32_fld_val** to save the contents of the FML or FML32 buffer. The following steps describe how to correlate statements using lrt_save_fld_val.

1. Insert the **lrt_save_fld_val** statement in your script where you want to save the contents of the current FML (or FML32) buffer.

   Example:

   ```
   lrt_save_fld_val (fbfr, "name", occurrence, "param_name");
   ```

2. Locate the **lrt** statements with the recorded values that you want to replace with the contents of the saved buffer. Replace all instances of the recorded values with the parameter name in curly brackets.

   Example:

   In the following example, a bank account was opened and the account number was stored to a parameter, **account_id**.

   ```
   /* Fill the data_0 buffer with new account information*/
   data_0 = lrt_tpalloc("FML", "", 512);
   lrt_Finitialize((FBFR*)data_0);
   lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=1", LRT_
   END_OF_PARMS);
   lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=S", LRT_
   END_OF_PARMS);
   ...
   LRT_END_OF_PARMS);
   lrt_Fadd_fld((FBFR*)data_0, "name=LAST_NAME", "value=Doe", ...);
   lrt_Fadd_fld((FBFR*)data_0, "name=FIRST_NAME", "value=John",
   ...);
   lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=234.12",
   ...);
   ```

```
/* Open a new account and save the new account number*/
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0,    &data_0,
&olen_2, 0);
lrt_abort_on_error();
lrt_save_fld_val((FBFR*)data_0, "name=ACCOUNT_ID", 0, "account_
id");
/* Use result from first query to fill buffer for the deposit*/
lrt_Finitialize((FBFR*)data_0);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCOUNT_ID", "value={account_
id}", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=200.11",
...);
```

In the above example, the account ID was represented by a field name, ACCOUNT_ID. Some systems represent a field by an ID number rather than a field name during recording.

You can correlate by field ID as follows:

```
lrt_save_fld_val((FBFR*)data_0, "id=8302", 0, "account_id");
```

Correlate Based on Buffer Location

This task describes how to correlate a string in a Tuxedo script by using the **lrt_save_parm** function. This function creates a correlation based on the location of a string within the buffer.

1. Insert an **lrt_save_parm** statement in your script at the point where you want to save the contents of the current buffer.
2. In the **replay.vdf** file, locate the buffer data that you want to replace with the contents of the saved buffer.

   View the buffer contents by selecting **replay.vdf** in the Action Pane of the main VuGen window (displayed by default in Script View).
3. Replace all instances of the value with the parameter name in curly brackets.

Example:

In the following example, an employee ID from a CARRAY buffer must be saved for later use. The recorded value was "G001" as shown in the output.

```
lrt_tpcall:227, PprLoad, 1782
Reply Buffer received.
...
123    "G001"
126    "..."
134    "Claudia"
```

Insert **lrt_save_parm** using the offset, 123, immediately after the request buffer that sends "PprLoad" and 227 bytes.

```
/* Request CARRAY buffer 57 */
    lrt_memcpy(data_0, buf_143, 227);
```

```
    tpresult_int = lrt_tpcall("PprLoad",
        data_0, 227, &data_1, &olen, TPSIGRSTRT);
    lrt_save_parm(data_1, 123, 9, "empid");
```

In the **replay.vdf** file, replace the recorded value, "G001", with the parameter, **empid**.

```
char buf_143[] =
"\xf5\x0\x0\x0\x4\x3\x2\x1\x1\x0\x0\x0\xbc\x2\x0\x0\x0\x0\x0\x0"
    "X"
"\x89\x0\x0\x0\xb\x0"
    "SPprLoadReq"
    "\xff\x0\x10\x0\x0\x4\x3\x6"
    "{empid}" // G001
    "\x7"
    "Claudia"
    "\xe"
    "LAST_NAME_SRCH"
...
```

This function can also be used to save a portion of a character array within an FML buffer. In the following example, the phone number is a character array, and the area code is the first three characters. First, the **lrt_save_fld_val** statement saves the phone number to a parameter, **phone_num**. The **lrt_save_parm** statement uses **lr_eval_string** to turn the phone number into a character array and then saves the area code into a parameter called **area_code**.

```
lrt_save_fld_val((FBFR*)data_0, "name=PHONE", 0, "phone_num");
lrt_save_parm(lr_eval_string("{phone_num}"), 0, 3, "area_code");
lr_log_message("The area code is %s\n", lr_eval_string("{area_
code}"));
```

Correlate Based on Delimiter

This task describes how to correlate a string in a Tuxedo script by using the **lrt_save_searched_string** function. This function creates a correlation based on the location of a delimiter in the buffer (e.g. correlate the string immediately after the first {). This function is recommended for PeopleSoft scripts because the reply buffers returned from the PeopleSoft server often differ in size during replay from what was seen during recording.

1. Insert an **lrt_save_searched_string** statement in your script where you want to save a portion of the current buffer.

   Note that the offset is the offset from the beginning of the string.

2. In the **replay.vdf** file, locate the buffer data that you want to replace with the contents of the saved buffer.

   View the buffer contents by selecting **replay.vdf** in the Action Pane of the main VuGen window (displayed by default in Script View).

3. Replace all instances of the value with the parameter name in curly brackets.

   Example:

In the following example, a Certificate is saved to a parameter for a later use. The **lrt_save_searched_string** function saves 16 bytes from the specified olen buffer, to the parameter **cert1**. The saved string location in the buffer, is 9 bytes past the first occurrence of the string "SCertRep".

This application is useful when the buffer's header information is different depending on the recording environment.

The certificate will come 9 bytes past the first occurrence of "SCertRep", but the length of the information before this string varies.

```
/* Request CARRAY buffer 1 */
lrt_memcpy(data_0, sbuf_1, 41);
lrt_display_buffer("sbuf_1", data_0, 41, 41);
data_1 = lrt_tpalloc("CARRAY", "", 8192);
tpresult_int = lrt_tpcall("GetCertificate",
    data_0,
    41,
    &data_1,
    &olen,
    TPSIGRSTRT);
/* Reply CARRAY buffer 1 */
lrt_display_buffer("rbuf_1", data_1, olen, 51);
lrt_abort_on_error();
lrt_save_searched_string(data_1, olen, 0, "SCertRep", 9, 16,
"cert1");
```

How to Correlate Scripts - Siebel Protocol

The following steps describe how to correlate Siebel Web Vuser scripts.

Correlation Library

To assist you with correlation, Siebel has released a correlation library file as part of the Siebel Application Server version 7.7. This library is available only through Siebel. The library file, **ssdtcorr.dll**, is located under the siebsrvr\bin folder for Windows and under siebsrvr/lib for Linux installations.

The library file, **ssdtcorr.dll**, must be available to all machines where a Load Generator or Controller reside. Support for this library requires VuGen 8.0 and higher. The following steps describe how to enable correlation with this library.

1. Copy the DLL file into the bin folder of the product installation.
2. Open a multi-protocol script using the **Siebel-Web** Vuser type.
3. Enable UTF-8 support in the **Recording Options > HTTP Properties > Advanced** node.
4. Open the recording option's Correlation node and click **Import**. Import the rules file, **WebSiebel77Correlation.cor**, from the \dat\webrulesdefaultsetting folder. If you are prompted with warnings, click **Override**.

   To revert to the default correlation, delete all of the Siebel rules and click **Use Defaults**.

When using the Siebel correlation library, verify that the SWE count rules (where the left boundary contains the **SWEC** string) are not disabled.

Correlation Rules

VuGen's native built-in rules for the Siebel server detect the Siebel server variables and strings, automatically saving them for use at a later point within the script. The rules list the boundary criteria that are unique for Siebel server strings.

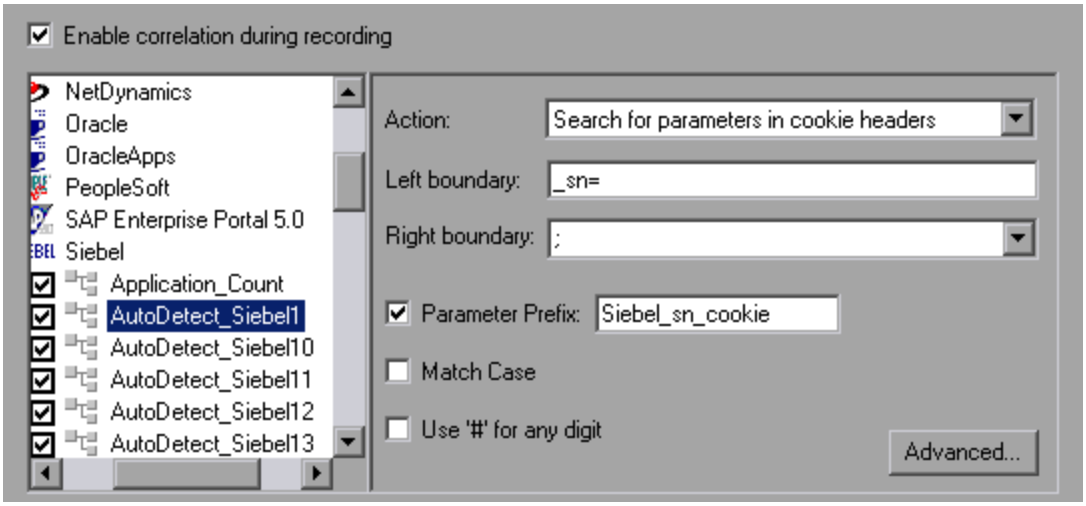
When VuGen detects a match using the boundary criteria, it saves the value between the boundaries to a parameter. The value can be a simple variable or a public function.

In normal situations, you do not need to disable any rules. In some cases, however, you may want to disable rules that do not apply. For example, disable Japanese content check rules when testing English-only applications.

Another reason to disable a rule is if the Controller explicitly requires an error condition to be generated. View the rule properties in the recording options and determine the conditions necessary for your application.

Simple Variable Correlation

In the following example, the left boundary criteria is `_sn=`. For every instance of `_sn=` in the left boundary and `;` in the right, VuGen creates a parameter with the **Siebel_sn_cookie** prefix.



In the following example, VuGen detected the _sn boundary. It saved the parameter to Siebel_sn_cookie6 and used it in the web_url function.

```
/* Registering parameter(s) from source
web_reg_save_param("Siebel_sn_cookie6",
"LB/IC=_sn=",
"RB/IC=;",
"Ord=1",
"Search=headers",
"RelFrameId=1",
LAST);
...
web_url("start.swe_3",
"URL=http://cannon.hplab.com/callcenter_
```

```
enu/start.swe?SWECmd=GotoPostedAction=;SWEDIC=true=;_sn={Siebel_sn_
cookie6}=;SWEC={Siebel_SWECount}=;SWEFrame=top._
sweclient=;SWECS=true",
"TargetFrame=",
"Resource=0",
"RecContentType=text/html",
"Referer=http://cannon.hplab.com/callcenter_
enu/start.swe?SWECmd=GetCachedFrame=;_sn={Siebel_sn_cookie6}=;SWEC=
{Siebel_SWECount}=;SWEFrame=top._swe",
"Snapshot=t4.inf",
"Mode=HTML",
LAST);
```

Function Correlation

In certain instances, the boundary match is a function. Functions generally use an array to store the run-time values. In order to correlate these values, VuGen parses the array and saves each argument to a separate parameter using the following format:

```
<parameter_name> = <recorded_value> (display_name)
```

The display name is the text that appears next to the value, in the Siebel Application.

VuGen inserts a comment block with all of the parameter definitions.

```
/* Registering parameter(s) from source task id 159
    // {Siebel_Star_Array_Op33_7} = ""
    // {Siebel_Star_Array_Op33_6} = "1-231"
    // {Siebel_Star_Array_Op33_2} = ""
    // {Siebel_Star_Array_Op33_8} = "Opportunity"
    // {Siebel_Star_Array_Op33_5} = "06/26/2003 19:55:23"
    // {Siebel_Star_Array_Op33_4} = "06/26/2003 19:55:23"
    // {Siebel_Star_Array_Op33_3} = ""
    // {Siebel_Star_Array_Op33_1} = "test camp"
    // {Siebel_Star_Array_Op33_9} = ""
    // {Siebel_Star_Array_Op33_rowid} = "1-6F"
    // */
```

In addition, when encountering a function, VuGen generates a new parameter for **web_reg_save_param**, **AutoCorrelationFunction**. VuGen also determines the prefix of the parameters and uses it as the parameter name. In the following example, the prefix is **Siebel_Star_Array_Op33**.

```
web_reg_save_param("Siebel_Star_Array_Op33",
    "LB/IC=`v`",
    "RB/IC=`",
    "Ord=1",
    "Search=Body",
    "RelFrameId=1",
    "AutoCorrelationFunction=flCorrelationCallbackParseStarArray",
    LAST);
```

VuGen uses the parameters at a later point within the script. In the following example, the parameter is called in **web_submit_data**.

web_submit_data("start.swe_14", "Action=http://cannon.hplab.com/callcenter_enu/start.swe", "Method=POST", "RecContentType=text/html", "Referer=", "Snapshot=t15.inf", "Mode=HTML", ITEMDATA, "Name=SWECLK", "Value=1", ENDITEM, "Name=SWEField", "Value=s_2_1_13_0", ENDITEM, "Name=SWER", "Value=0", ENDITEM, "Name=SWESP", "Value=false", ENDITEM, "Name=s_2_2_29_0", "Value={**Siebel_Star_Array_Op33_1**}", ENDITEM, "Name=s_2_2_30_0", "Value={**Siebel_Star_Array_Op33_2**}", ENDITEM, "Name=s_2_2_36_0", "Value={**Siebel_Star_Array_Op33_3**}", ENDITEM, ...

During replay, Vusers do a callback to the public function, using the array elements that were saved as parameters.

Note: Correlation for the **SWEC** parameter is not done through the correlation rules. VuGen handles it automatically with a built-in detection mechanism. For more information, see SWEC Correlation.

SWEC Correlation

SWEC is a parameter used by Siebel servers representing the number of user clicks. The SWEC parameter usually appears as an argument of a URL or a POST statement. For example:

```
GET /callcenter_enu/start.swe?SWECmd=GetCachedFrame=;_sn=2-
mOTFXHWBAAGb5Xzv9Ls2Z45QvxGQnOnPVtX6vnfUU_=;SWEC=1=;SWEFrame=top._
swe._sweapp HTTP/1.1
```

or

```
POST /callcenter_enu/start.swe HTTP/1.1
...
\r\n\r\n
SWERPC=1=;SWEC=0=;_sn=2-mOTFXHWBAAGb5Xzv9Ls2Z45QvxGQnOnPVtX6vnfUU_
=;SWECmd=InvokeMethod...
```

VuGen handles the changes of the SWEC by incrementing a counter before each relevant step. VuGen stores the current value of the SWEC in a separate variable (Siebel_SWECount_var). Before each step, VuGen saves the counter's value to a VuGen parameter (Siebel_SWECount).

In the following example, web_submit_data uses the dynamic value of the SWEC parameter, Siebel_SWECount.

```
Siebel_SWECount_var += 1;
lr_save_int(Siebel_SWECount_var, "Siebel_SWECount");
web_submit_data("start.swe_8",
    "Action=http://cannon.hplab.com/callcenter_enu/start.swe",
    "Method=POST",
    "TargetFrame=",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t9.inf",
    "Mode=HTML",
    "EncodeAtSign=YES",
    ITEMDATA,
    "Name=SWERPC", "Value=1", ENDITEM,
    "Name=SWEC", "Value={Siebel_SWECount}", ENDITEM,
```

```
"Name=SWECmd", "Value=InvokeMethod", ENDITEM,
"Name=SWEService", "Value=SWE Command Manager", ENDITEM,
"Name=SWEMethod", "Value=BatchCanInvoke", ENDITEM,
"Name=SWEIPS",...
LAST);
```

Note that the SWEC parameter may also appear in the referrer URL. However, its value in the referrer URL usually differs from its value in the requested URL. VuGen handles this automatically.

Correlate SWECount Parameters

The SWECount parameter value is usually a small number consisting of one or two digits. It is often difficult to determine where to replace the recorded value with a parameter.

In the **web_submit_data** function, VuGen only replaces it in the SWEC field.

In URLs, VuGen only replaces the value when it appears after the strings "SWEC=" or "SWEC`".

The parameter name for all the SWECount correlations is the same.

Correlate ROWID Parameters

In certain cases, the **rowid** is preceded by its length, encoded in hexadecimal format. Since this length can change, this value must be correlated.

For example, the following string is comprised of a length value and RowID, `xxx6_1-4ABCyyy`, where 6 is the length, and `1-4ABC` is the RowID.

If you define parameters to correlate the string as

```
xxx{rowid_Length}_{rowid}yyy
```

then using this enhanced correlation, VuGen generates the following function before the string:

```
web_save_param_length("rowid", LAST);
```

This function gets the value of **rowid**, and saves its length into the parameter **rowid_length** in hexadecimal format.

Correlate SWET (timestamp) Parameters

The SWETS value in the script, is the number of milliseconds since midnight January 1st, 1970.

VuGen replaces all non-empty timestamps in the script, with the parameter {SiebelTimeStamp}. Before saving a value to this parameter, VuGen generates the following function:

```
web_save_timestamp_param("SiebelTimeStamp", LAST);
```

This function saves the current timestamp to the **SiebelTimeStamp** parameter.

How to Correlate Scripts - Oracle NCA

The following steps describe different items in Oracle NCA scripts that may need correlation.

Correlate Statements for Load Balancing

VuGen supports load balancing for multiple application servers. You correlate the HTTP return values with the **nca_connect_server** parameters. The Vuser then connects to the relevant server

during test execution, applying load balancing. The following steps describe how to correlate statements for load balancing.

1. **Record a multi-protocol script.**

   Record a multi-protocol script for Oracle NCA and Web Protocols. Perform the desired actions and save the script.

2. **Define parameters for host and host arguments.**

   Define two variables, **serverHost** and **serverArgs**, for parameterization:

```
web_set_max_html_param_len("512");
web_reg_save_param("serverHost", "NOTFOUND=ERROR",
    "LB=<PARAM name=\"serverHost\" value=\"","RB=\">", LAST);
web_reg_save_param("serverArgs", "NOTFOUND=ERROR",
    "LB=<PARAM name=\"serverArgs\" value=\"","RB=\">", LAST);
```

3. **Assign values to serverHost and serverArgs:**
```
web_url("step_name", "URL=http://server1.acme.com/test.htm", LAST);
```

4. **Modify the nca_connect_server statement from:**
```
nca_connect_server("199.203.78.170",
                   9000"/*version=107*/,
"module=e:\\appsnca...fndnam=apps ");
```

   to:

```
nca_connect_server("{ serverHost }", "9000"/*version=107*/, "
{serverArgs}");
```

The script should now look like this:

```
web_set_max_html_param_len("512");
web_reg_save_param("serverHost", "NOTFOUND=ERROR",
    "LB=<PARAM name=\"serverHost\" value=\"","RB=\">", LAST);
web_reg_save_param("serverArgs", "NOTFOUND=ERROR",
    "LB=<PARAM name=\"serverArgs\" value=\"","RB=\">", LAST);
    web_url("step_name", "URL=http://server1.acme/test.htm", LAST);
nca_connect_server("{serverHost}","9000"/*version=107*/,"{serverArgs}
");
```

Correlate the icx_ticket Variable

The icx_ticket variable, is part of the information sent in the **web_url** and **nca_connect_server** functions:

```
web_url("fnd_icx_launch.runforms",
"URL=http://ABC-123:8002/pls/VIS/fnd_icx_launch.runforms\?ICX_
TICKET=5843A55058947ED3=;RESP_APP=AR=;RESP_KEY=RECEIVABLES_
MANAGER=;SECGRP_KEY=STANDARD", LAST);
```

This **icx_ticket** value is different for each recording. It contains cookie information sent by the client. To correlate your recording, add **web_reg_save_param** before the first occurrence of the recorded **icx_ticket** value:

```
web_reg_save_param("icx_ticket", "LB=TICKET=", "RB==;RES", LAST);
...
```

```
web_url("fnd_icx_launch.runforms",
"URL=http://ABC-123:8002/pls/VIS/fnd_icx_launch.runforms\?ICX_TICKET=
{icx_ticket}=;RESP_APP=AR=;RESP_KEY=RECEIVABLES_MANAGER=;SECGRP_
KEY=STANDARD", LAST);
```

Note: The left and right boundaries of **web_reg_save_param** may differ depending on your application setup.

Correlate the JServSessionIdroot Values

The **JServSessionIdroot** value is a cookie that the application sets to store the session ID. In most cases, VuGen automatically correlates this value and inserts a **web_reg_save_param** function. If VuGen did not add this function automatically, you add it manually, replacing all of its occurrences with the parameter name.

To identify the value that you need to correlate, open the Execution log (**View > Output Window**) and locate the response body.

```
vuser_init.c(8):     Set-Cookie: JServSessionIdroot=my1sanw2n1.JS4;
path=/\r\n
vuser_init.c(8):     Content-Length: 79\r\n
vuser_init.c(8):     Content-Type: text/plain\r\n
vuser_init.c(8):     \r\n
vuser_init.c(8):     81-byte response body for "http://ABC-
123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo=;
                          ifhost=mercury=;ifip=123.45.789.12"
(RelFrameId=1)
vuser_init.c(8):
/ser-
vlet/or-
acle.forms.servlet.ListenerServlet?JServSessionIdroot=my1sanw2n1.JS4\r\n
```

To correlate this dynamic value, insert a **web_reg_save_param** function before the first occurrence and then replace the variable value with the parameter name throughout the script. In this example, the right and left boundaries are \r and \n, but you should check your specific environment to determine the exact boundaries in your environment.

```
web_reg_save_param("NCAJServSessionId","LB=\r\n\r\n","RB=\r","ORD=1",
LAST);
web_url("f60servlet",
    "URL= http://ABC-
"123/servlet/oracle.forms.servlet.ListenerServlet?ifcmd=getinfo=;"
"ifhost=mercury=;ifip=123.45.789.12", LAST);
web_url("oracle.forms.servlet.ListenerSer",
    "URL=http://ABC-123{NCAJServSessionId}?ifcmd=getinfo=;"
"ifhost=mercury=;ifip=123.45.789.12", LAST);
```

How to Correlate Scripts - Microsoft .NET

This task describes how to correlate Microsoft .NET Vuser scripts.

Correlate Scripts Using ADO.net Environments

1. **Locate the dataset in your script.**

   Display the Vuser script in the Editor and expand the applicable **DATASET_XML** statement. Click **View > Snapshot**.

	CustomerID	CompanyName	ContactName	ContactTitle	Address
1	ABC	ABC Company	John Smith	Owner	One My Way
2	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
3	ANATR	Ana Trujillo Emparec	Ana Trujillo	Owner	Avda. de la C
4	ANTON	Antonio Moreno Tac	Antonio Moreno	Owner	Mataderos 2
5	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover

2. **Locate the value.**

   Locate the value you want to correlate. To search for a value in a data grid, display the data grid in the Snapshot pane, and click **Search > Quick Find** to open the Search dialog box.

   In the Search dialog box, click **Include in Search**, and then select the **Snapshots** check box.

3. **Create a correlation.**

   Right-click on the value in the grid that you want to correlate and select **Create Correlation**. The Create a correlation dialog box opens.

4. **Specify a parameter name.**

   Specify a parameter name, identical to the variable you defined earlier. Click **OK**. VuGen prompts you if you want to search for all occurrences. Click **OK**.

   VuGen adds an **lr.save_string** function before each dataset. For example:

```
lr.save_string("MyCustomerID", CustomerAndOrdersDataSet_3.Tables
["Customers"].Rows[0]["CompanyName"].ToString());
```

5. **Reference the parameter at a later point in the script.**

   Select the value you want to replace with a parameter and select **Replace with a parameter** from the right-click menu. Insert the saved variable name in the Parameter name box. Click OK. VuGen prompts you to replace all of the values with a parameter, using the **lr.eval_string** function to evaluate the string's value.

```
lr.message("The customer ID is""+ lr.eval_string("{MyCustomerID}") +
");
```

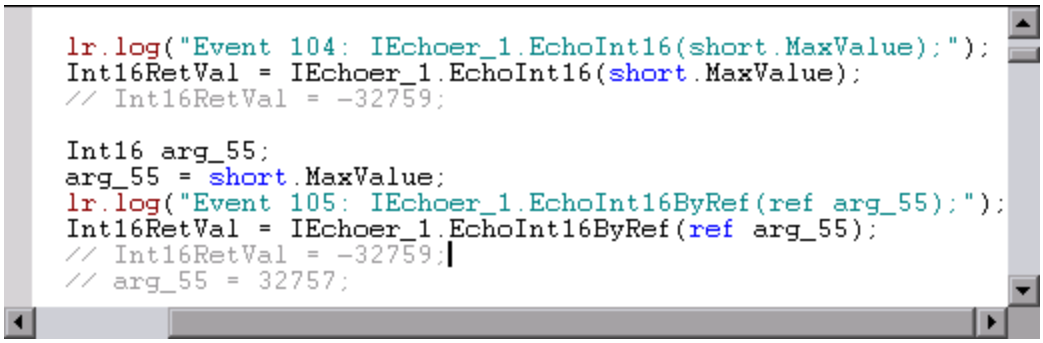
Unlike other protocols, the script includes direct calls to the application or framework method. Therefore, you cannot replace the string value with the {paramName}—instead you must use **lr.eval_string** to evaluate the parameter's value.

Correlate with Output Parameters

For primitive values, you should generate the script with output parameter values and examine the output parameters for correlations.

1. Select **Recording > Recording Options**, and select the **General > Script** node.
2. Select the **Insert output parameter values** check box. Click **OK** to close the Recording Options dialog box.

3. Select **Record > Regenerate Script** to regenerate the script.
4. Search the commented output primitive values for correlations.



For more information about using correlation functions, see the Function Reference (**Help > Function Reference**).

How to Correlate Scripts - Java Scripts - Serialization

In RMI, and some cases of CORBA, the client AUT creates a new instance of a Java object using the **java.io.serializable** interface. It passes this instance as a parameter for a server invocation. In the following segment, the instance **p** is created and passed as a parameter.

```
// AUT code:
java.awt.Point p =  new java.awt.Point(3,7);
map.set_point(p);
:
```

The automatic correlation mechanism is ineffective here, since the object did not return from any previous call. In this case, VuGen activates the serialization mechanism and stores the object being passed as a parameter. It saves the information to a binary data file under the user folder. Additional parameters are saved as new binary data files, numbered sequentially. VuGen generates the following code:

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
          java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
          map.set_point(p);
      }
:
}
```

The integer passed to **lr.deserialize** represents the number of binary data files in the Vuser folder.

To parameterize the recorded value, use the public setLocation method (for information, see the JDK function reference). The following example uses the **setLocation** method to set the value of the object, p.

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
          java.awt.Point p = (java.awt.Point)lr.deserialize(0, false);
```

```
            p.setLocation(2,9);
            map.set_point(p);
        }
:
:
}
```

In certain instances the public method of **setLocation** is not applicable. As an alternative, you can use the API of your class that incorporate get or set accessor methods. If you are working with AUT classes that do not have get/set methods or use private methods, or if you are unfamiliar with the classes' API, you can use VuGen's built-in serialization mechanism. This mechanism allows you to expand objects in their ASCII representation and manually parameterize the script. You enable this mechanism in the Recording Options dialog box. For details, see "Recording Properties > Serialization Options Node" on page 324.

VuGen generates an **lr.deserialize** method that deserializes the data or displays complex data structures as serial strings. Once the structure is broken down to its components, it is easier to parameterize. The **lr.deserialize** method receives two arguments, a string and an integer. The string is the parameter's value that is to be substituted during replay. The integer is the index number of binary file to load.

If you choose not to expand objects in your script by clearing the Unfold Serialized Objects check box, you can control the serialization mechanism by passing arguments to the lr.deserialize method. The first argument is an integer indicating the number of binary files to load. The second integer is a boolean value:

true	Use VuGen's serialization mechanism.
false	Use the standard Java serialization mechanism.

The following segment shows a generated script in which the serialization mechanism was enabled.

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
            _string = "java.awt.Point __CURRENT_OBJECT = {" +
                      "int x = "#5#" +
                      "int y = "#8#" +
            "}";
            java.awt.Point p = (java.awt.Point)lr.deserialize(_string,
0);
            map.set_point(p);
        }
:
}
```

The string values are placed between delimiters. The default delimiter is "#". You can change the delimiter in the **Serialization** tab of the recording options. Delimiters are used to speed up the parsing of the string during replay.

When modifying the string, you must maintain the following rules:

- Order of lines may not be changed. The parser reads the values one-by-one—not the member names.
- Only values between two delimiters may be modified.
- Object references may not be modified. Object references are indicated only to maintain internal consistency.
- "_NULL_" can appear as a value, representing the Java null constant. You can replace it with string type values only.
- Objects may be deserialized anywhere in the script. For example, you can deserialize all objects in the **init** method and use the values in the **action** method.
- Maintain internal consistency for the objects. For example, if a member of a vector is **element count** and you add an element, you must modify the element count.

In the following segment, a vector contains two elements:

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
          _string = "java.util.Vector CURRENTOBJECT = {" +
                    "int capacityIncrement = "#0#" +
                    "int elementCount = #2#" +
                    "java/lang/Object elementData[] = {" +
                           "elementData[0] = #First Element#" +
                           "elementData[1] = #Second Element#" +
                           "elementData[2] = _NULL_" +
                            ....
                           "elementData[9] = _NULL_" +
                    "}" +
          "}";
          _vector = (java.util.Vector)lr.deserialize(_string,0);
          map.set_vector(_vector);
      }
:
}
```

In the following example, one of the vector's elements was changed—a "_NULL_" value was changed to "Third element". In coordination with the addition of the new element, the **elementCount** member was modified to **3**.

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
          _string = "java.util.Vector CURRENTOBJECT = {" +
                    "int capacityIncrement = "#0#" +
                    "int elementCount =  #3# " +
                    "java/lang/Object elementData[] = {" +
                           "elementData[0] = #First Element#" +
                           "elementData[1] = #Second Element#" +
                           "elementData[2] =  #Third Element#" +
                             ....
```

```
                                "elementData[9] = _NULL_" +
                        "}" +
            "}";
            _vector = (java.util.Vector)lr.deserialize(_string,0);
            map.set_vector(_vector);
        }
:
}
```

Due to the complexity of the serialization mechanism, which opens up the objects to ASCII representation, opening large objects while recording may increase the time required for script generation. To decrease this time, you can specify flags which will improve the performance of the serialization mechanism.

When adding **lr.deserialize** to your script, we recommend that you add it to the **init** method—not the **action** method. This will improve performance since VuGen will only deserialize the strings once. If it appears in the **action** method, VuGen will deserialize strings for every iteration.

How to Correlate Scripts - Java

VuGen's Java recorder attempts to automatically correlate statements in the generated script. It performs correlation on Java objects only. When it encounters a Java primitive (byte, character, boolean, integer, float, double, short, and long) during recording, the argument values appear in the script without association to variables. VuGen automatically correlates all objects, arrays of objects, and arrays of primitives. Note that Java arrays and strings are also considered objects.

VuGen employs several levels of correlation: Standard, Enhanced, Strings. You enable or disable correlation from the Recording options. An additional method of Serialization can be used to handle scripts where none of the former methods can be applied.

Standard Correlation

Standard correlation refers to the automatic correlation performed during recording for simple objects, excluding object arrays, vectors, and container constructs.

When the recorded application invokes a method that returns an object, VuGen's correlation mechanism records these objects. When you run the script, VuGen compares the generated objects to the recorded objects. If the objects match, the same object is used. The following example shows two CORBA objects my_bank and my_account. The first object, my_bank, is invoked; the second object, my_account, is correlated and passed as a parameter in final line of the segment:

```
public class Actions {
  // Public function: init
      public int init() throws Throwable {
       Bank my_bank = bankHelper.bind("bank", "shunra");
       Account my_account = accountHelper.bind("account","shunra");
       my_bank.remove_account(my_account);
      }
:
}
```

Advanced Correlation

Advanced or **deep** correlation refers to the automatic correlation performed during recording for complex objects, such as object arrays and CORBA container constructs.

The deep correlation mechanism handles CORBA constructs (structures, unions, sequences, arrays, holders, `any's) as containers. This allows it to reference inner members of containers, additional objects, or different containers. Whenever an object is invoked or passed as a parameter, it is also compared against the inner members of the containers.

In the following example, VuGen performs deep correlation by referencing an element of an array. The remove_account object receives an account object as a parameter. During recording, the correlation mechanism searches the returned array my_accounts and determines that its sixth element should be passed as a parameter.

```
public class Actions {
   // Public function: init
   public int init() throws Throwable {
      my_banks[] = bankHelper.bind("banks", "shunra");
      my_accounts[] = accountHelper.bind("accounts","shunra");
      my_banks[2].remove_account(my_accounts[6]);
    }
:
}
```

The following segment further illustrates enhanced correlation. The script invokes the send_letter object that received an address type argument. The correlation mechanism retrieves the inner member, address, in the sixth element of the my_accounts array.

```
public class Actions {
   // Public function: init
   public int init() throws Throwable {
      my_banks = bankHelper.bind("bank", "shunra");
      my_accounts = accountHelper.bind("account", "shunra");
      my_banks[2].send_letter(my_accounts[6].address);
    }
:
}
```

String Correlation

String correlation refers to the representation of a recorded value as an actual string or a variable. When you disable string correlation (the default setting), the actual recorded value of the string is indicated explicitly within the script. When you enable string correlation, it creates a variable for each string, allowing you to use it at a later point in the script.

In the following segment, string correlation is enabled—you store the value returned from the get_id method in a string type variable for use later on in the script.

```
public class Actions {
   // Public function: init
   public int init() throws Throwable {
      my_bank = bankHelper.bind("bank", "shunra");
      my_account1 = accountHelper.bind("account1", "shunra");
      my_account2 = accountHelper.bind("account2", "shunra");
      string = my_account1.get_id();
```

```
        string2 = my_account2.get_id();
        my_bank.transfer_money(string, string2);
    }
:
}
```

How to Correlate Scripts - XPath Correlation in Flex Vuser Scripts

This topic describes how to use XPath correlation in Flex Vuser scripts. You use the XML View inside the Snapshot pane to perform the correlation. Before you can successfully implement XPath correlation, you must first configure the recording options.

For details on how to use regular correlation in Flex Vuser scripts, see "How to Correlate Scripts Using Design Studio" on page 137.

1. Configure the recording options.
   a. Select **Record > Recording Options**.
   b. Under **Flex**, click **Externalizable Objects**.
   c. Click **Serialize objects using**, and select **Custom Java Classes**.
   d. Click the **Add jar or zip file** button  .
   e. On the LiveCycle installation discs, locate the following three files, and add them to the **Classpath Entries** list:
      i. **flex.jar**
      ii. **flex-messaging-common.jar**
      iii. **flex-messaging-core.jar**

   Ensure that the added files exist in the same location on all load generator machines.
2. Record the Vuser script.
3. In the Editor, click inside the **flex_amf_call** step that contains the data you want to correlate, or in the Step Navigator, double-click the **flex_amf_call** step that contains the data you want to correlate.
4. Click **View > Snapshot** or click the **Snapshot** button  on the VuGen toolbar.
5. In the Snapshot pane, click the **Response Body** tab.
6. On the right-side of the Snapshot pane, click the **XML View** tab.
7. In the XML View, locate and select the entire string that contains the dynamic data that requires correlation.
8. Right-click inside the selection, and select **Create Correlation**. The Design Studio opens. For details on how to use Design Studio, see "Correlation Tab [Design Studio] Overview" on page 134.

When the correlation is complete, VuGen adds a **web_reg_save_parm_xpath** step to the Vuser script.

How to Correlate Scripts - COM Protocol

The following steps describe how to correlate COM Vuser scripts.

1. Select **View** > **Output** to display the output tabs at the bottom of the window. Check for errors in the Replay tab. Often, these errors can be corrected by correlation.
2. Select the relevant step in the Step Navigator, and view the step in the **Snaphot pane**.
3. Right click the value in the snapshot and select **Create correlation**. This will open the **Design Studio** window.
4. Select the value you would like to correlate by highlighting it in the grid and clicking the **Correlate** button.

   When a value is correlated, VuGen adds the correlation parameter and saves the original value in a comment in the script.

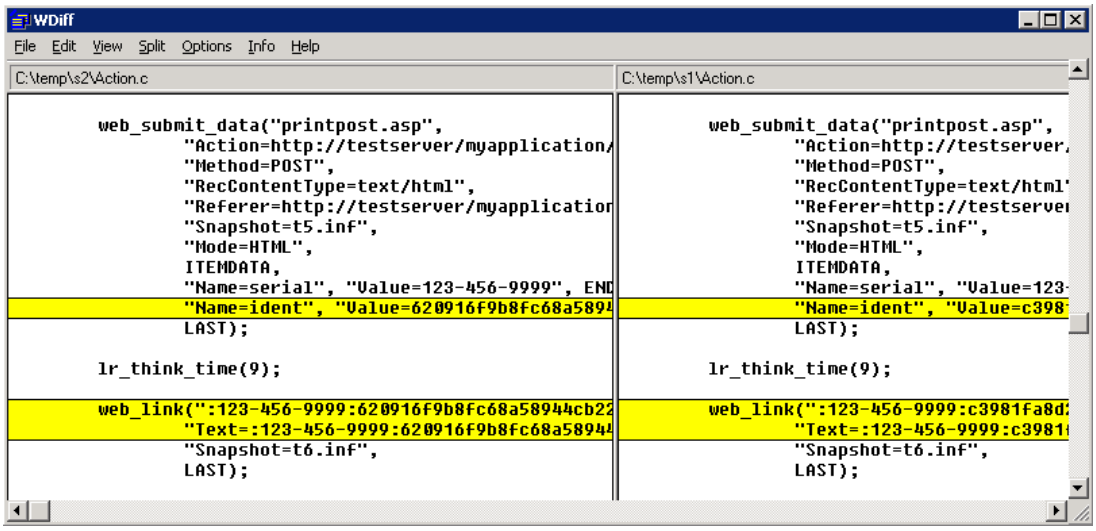
```
252 /* Correlation comment - Do not change!  Original value='1' Name ='CorrelationParameter' Type ='Manual' */
253     lrc_save_rs_param(_Recordset_45,
254         1,
255         2,
256         0,
257         "CorrelationParameter1");
```

How to Search for Values that Need Correlation

The following steps describe different ways to search for values that need correlation.

Search by Comparing Scripts

1. Record a script and save it.
2. Create a new script and record the identical operations. Save the script.
3. Select **Tools** > **Compare with Vuser** to compare the scripts. For more details, see "How to Compare Scripts Side by Side" on page 60.
4. Differences in the script are highlighted. Review the differences to determine which ones may require correlation.



Note: *WDiff* is the default utility, but you can specify a custom comparison tool. For more information, see "How to Compare Scripts Side by Side" on page 60.

Replay Log Search

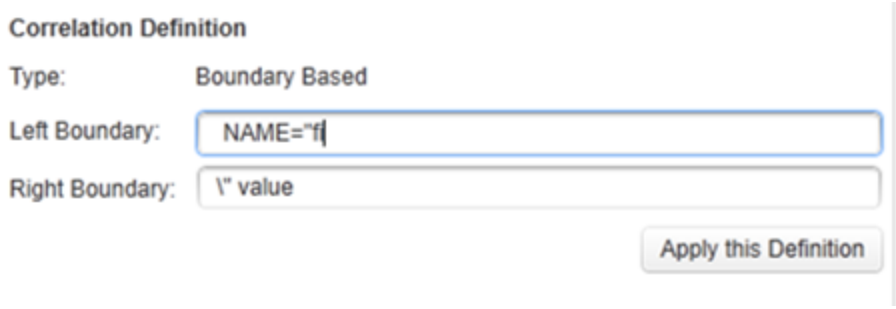
1. Scan the script in script view for strings that may need correlation such as hash strings, random strings, session ID's, and so on.
2. Search the generation log for the first time that the string appears (this is the response from the server).
3. Search the extended replay log for the same response. Check to see if this response contains a different string within the same boundaries as the original suspected string. If yes, this string requires correlation.

How to Modify Correlation Definitions

You can modify correlation definitions to help eliminate dynamic values that do not require correlation.These tasks describe how to modify boundary based, regular expression, and XPath query correlation definitions for record or response correlation.

Modifying Boundary Based Correlation Definitions

1. Click the **Design Studio** button on the VuGen toolbar.
2. Select a dynamic value from the correlation grid and expand **Details**.
3. Edit the **Left Boundary** or **Right Boundary** under the **Correlation Definition** section. You can modify the definition by adding or deleting text.



4. Click **Apply this Definition**.

   The **Apply this Definition** button will not be enabled unless the modified boundary definition occurs in the snapshot and the script.

   Note: If you do not apply the definition before selecting another dynamic value in the grid, your changes will be lost. If you select **Replay & Scan** before correlating your value with the modified definition, your changes will be lost.

Modifying Regular Expression Correlation Definitions

1. Click the **Design Studio** button on the VuGen toolbar.
2. Select a dynamic value from the correlation grid and expand **Details**.

3. Edit the **Regular Expression** under the **Correlation Definition** section.
4. Click **Apply this Definition**.

   The **Apply this Definition** button will not be enabled unless the modified boundary definition occurs in the snapshot and the script.

   Note: If you do not apply the definition before selecting another dynamic value in the grid, your changes will be lost. If you select **Replay & Scan** before correlating your value with the modified definition, your changes will be lost.

Modifying XPath Correlation Definitions

1. Click the **Design Studio** button on the VuGen toolbar.
2. Select a dynamic value from the correlation grid and expand **Details**.
3. Edit the XPath definition under the **Correlation Definition** section.
4. Click **Apply this Definition**.

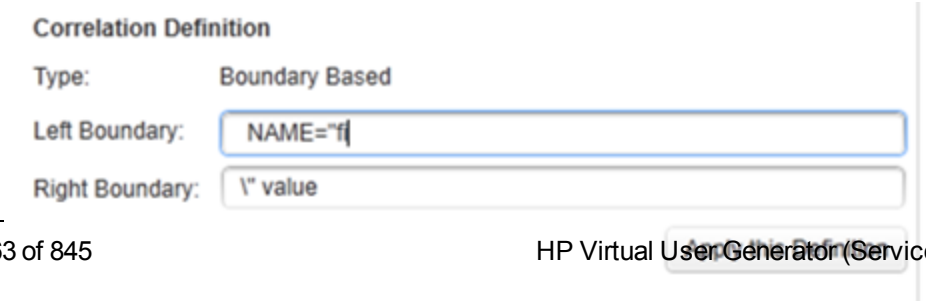
   The **Apply this Definition** button will not be enabled unless the modified boundary definition occurs in the snapshot and the script.

   Note: If you do not apply the definition before selecting another dynamic value in the grid, your changes will be lost. If you select **Replay & Scan** before correlating your value with the modified definition, your changes will be lost.

Modifying Winsocket Correlation Definitions

1. Winsocket dynamic values are correlated from the snapshot. To access, select the relevant step in the **Step Navigator**, and view the step in the **Snaphot pane**. The Winsocket protocol has both a hex and text snapshot.

   Right click the value in the snapshot and select **Create correlation** or **Create boundary correlation**. This will open the **Design Studio**window.
2. Select a dynamic value from the correlation grid and expand **Details**.
3. If you selected **Create correlation**, edit the Data Range in the **Correlation Definition** section. If you selected **Create boundary correlation**, edit the left of right boundary.

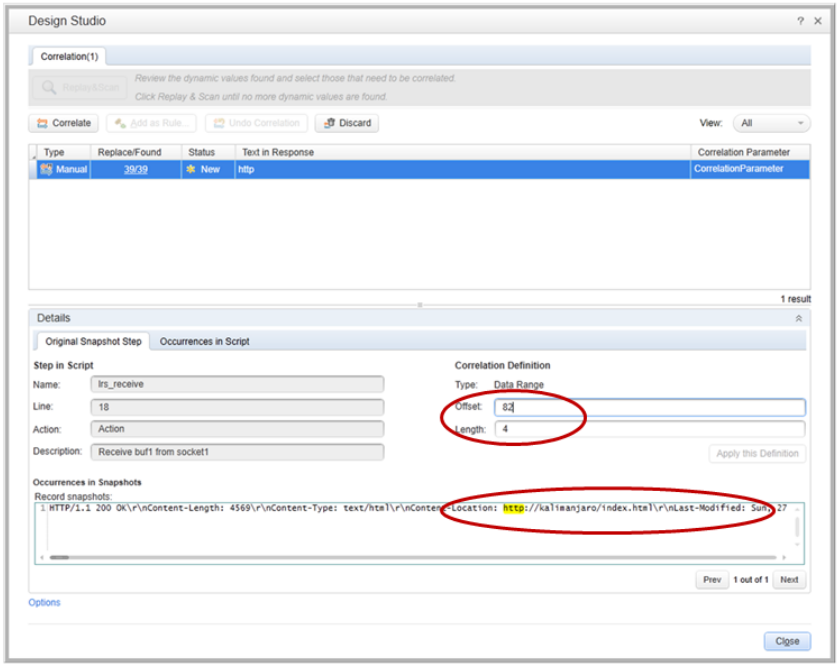


4. Click **Apply this Definition**.

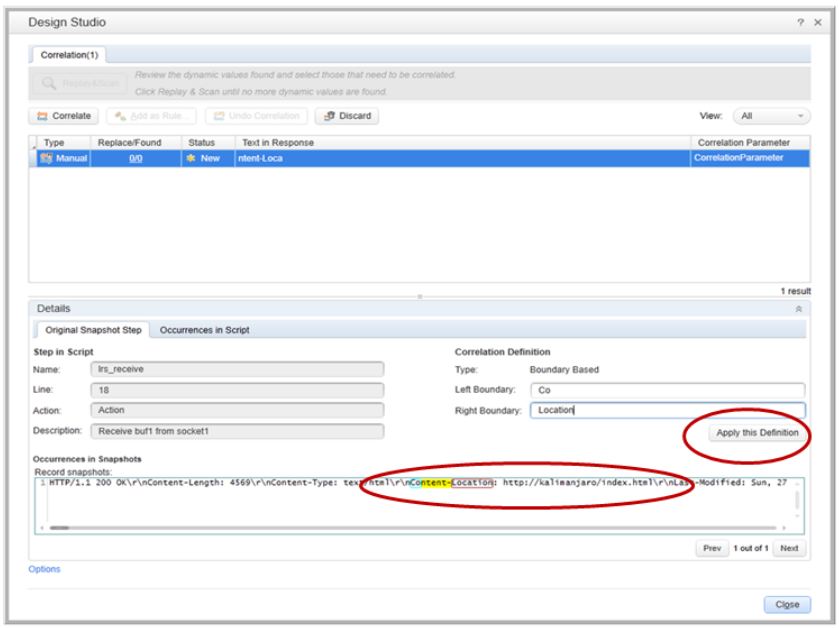
   The **Apply this Definition** button will not be enabled unless the modified boundary definition occurs in the snapshot.

   View the following images that display both a Data Range definition and a Boundary definition.

Data Range definition correctly modified



Boundary base definition correctly modified



Note: If you do not apply the definition before selecting another dynamic value in the grid, your changes will be lost. If you select **Replay & Scan** before correlating your value with the modified definition, your changes will be lost.

Note: Correlation Rule definitions cannot be modified using this method.

Correlation Functions - Database Vuser Scripts

When working with Database Vuser scripts, (DbLib, CtLib, Oracle, and so forth) you can use VuGen's automated correlation feature to insert the appropriate functions into your script. The correlating functions are:

- **lrd_save_col** saves a query result appearing in a grid, to a parameter. This function is placed before fetching the data. It assigns the value retrieved by the subsequent **lrd_fetch** to the specified parameter.
(**lrd_ora8_save_col** for Oracle 8 and higher)
- **lrd_save_value** saves the current value of a placeholder descriptor to a parameter. It is used with database functions that set output placeholders (such as certain stored procedures under Oracle).
- **lrd_save_ret_param** saves a stored procedure's return value to a parameter. It is used primarily with database procedures stored in DbLib that generate return values.

Note: VuGen does not apply correlation if the saved value is invalid or NULL (no rows returned).

For more information about these functions and their arguments, see the Function Reference (**Help > Function Reference**).

Correlation Functions - Java Vuser Scripts

To correlate statements for Java Vusers, you can use the Java Vuser correlation functions. These functions may be used for all Java type Vusers, to save a string to a parameter and retrieve it when required.

lr.eval_string	Replaces a parameter with its current value.
lr.eval_data	Replaces a parameter with a byte value.
lr.eval_int	Replaces a parameter with an integer value.
lr.eval_string	Replaces a parameter with a string.

lr.save_data	Saves a byte as a parameter.
lr.save_int	Saves an integer as a parameter.
lr.save_string	Saves a null-terminated string to a parameter.

When recording a CORBA or RMI session, VuGen performs correlation internally. For more information, see "How to Correlate Scripts - Java" on page 158.

Using the Java String Functions

When programming Java Vuser scripts, you can use the Java Vuser string functions to correlate your scripts. In the following example, **lr.eval_int** substitutes the variable **ID_num** with its value, defined at an earlier point in the script.

```
lr.message(" Track Stock: " + lr.eval_int(ID_num));
```

In the following example, **lr.save_string** assigns John Doe to the parameter Student. This parameter is then used in an output message.

```
lr.save_string("John Doe", "Student");
// ...
lr.message("Get report card for " + lr.eval_string("<Student>"));
classroom.getReportCard
```

Web_reg_save_param function details

When you run a script, the **web_reg_save_param** function scans the subsequent HTML page that is accessed. You specify a left and/or right boundary and VuGen searches for text between those boundaries. When VuGen finds the text, it assigns it to a parameter.

The function's syntax is as follows:

```
int web_reg_save_param (const char *mpszParamName, <List of
Attributes>, LAST);
```

The following table lists the available attributes. Note that the attribute value strings (for example, Search=all) are not case sensitive.

NotFound	The handling method when a boundary is not found and an empty string is generated. "ERROR," the default, indicates that VuGen should issue an error when a boundary is not found. When set to "EMPTY," no error message is issued and script execution continues. Note that if Continue on Error is enabled for the script, then even when NOTFOUND is set to "ERROR," the script continues when the boundary is not found, but it writes an error message to the Extended log file.
LB	The left boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.

RB	The right boundary of the parameter or the dynamic data. This parameter must be a non-empty, null-terminated character string. Boundary parameters are case sensitive; to ignore the case, add "/IC" after the boundary. Specify "/BIN" after the boundary to specify binary data.
RelFrameID	The hierarchy level of the HTML page relative to the requested URL. The possible values are ALL or a number.
Search	The scope of the search—where to search for the delimited data. The possible values are Headers (search only the headers), Body (search only body data, not headers), or ALL (search body and headers). The default value is ALL.
ORD	This optional parameter indicates the ordinal or occurrence number of the match. The default ordinal is 1. If you specify "All," it saves the parameter values in an array.
SaveOffset	The offset of a sub-string of the found value, to save to the parameter. The default is 0. The offset value must be non-negative.
SaveLen	The length of a sub-string of the found value, from the specified offset, to save to the parameter. The default is -1, indicating until the end of the string.
Convert	The conversion method to apply to the data: HTML_TO_URL: convert HTML-encoded data to a URL-encoded data format HTML_TO_TEXT: convert HTML-encoded data to plain text format

Correlation Functions - C Vuser Scripts

To correlate statements for protocols that do not have specific functions, you can use the C Vuser correlation functions. These functions can be used for all C-type Vusers, to save a string to a parameter and retrieve it when required.

lr_eval_string	Replaces all occurrences of a parameter with its current value.
lr_save_string	Saves a null-terminated string to a parameter.
lr_save_var	Saves a variable length string to a parameter.

For additional information about the syntax of these functions, see the Function Reference (**Help > Function Reference**).

Using lr_eval_string

In the following example, lr_eval_string replaces the parameter row_cnt with its current value. This value is sent to the Output window using lr_output_message.

```
lrd_stmt(Csr1, "select count(*) from employee", -1, 1 /*Deferred*/,
...);
lrd_bind_col(Csr1, 1, =;COUNT_D1, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_save_col(Csr1, 1, 1, 0, "row_cnt");
```

```
lrd_fetch(Csr1, 1, 1, 0, PrintRow2, 0);
lr_output_message("value: %s" , lr_eval_string("The row count is:
<row_cnt>"));
```

Using lr_save_string

To save a NULL terminated string to a parameter, use **lr_save_string**. To save a variable length string, use **lr_save_var** and specify the length of the string to save.

In the following example, lr_save_string assigns 777 to a parameter emp_id. This parameter is then used in another query or for further processing.

```
lrd_stmt(Csr1, "select id from employees where name='John'",...);
lrd_bind_col(Csr1,1,=;ID_D1,...);
lrd_exec(Csr1, ...);
lrd_fetch(Csr1, 1, ...);
/* GRID showing returned value "777" */
lr_save_string("777", "emp_id");
```

Design Studio [Correlation Tab] Dialog Box

This dialog box enables you to scan for, correlate, and view information about dynamic values in your script.

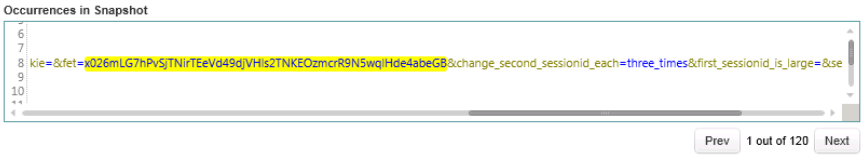
To access	Click the **Design Studio** button on the VuGen toolbar. The button is enabled only when you have a recorded script in the Solution Explorer.
Important information	"Correlation Tab [Design Studio] Overview" on page 134
Relevant tasks	"How to Correlate Scripts Using Design Studio" on page 137

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Correlation Tab	
Replay and Scan	Design studio scans for dynamic values using all enabled types: rule, record, and replay.
Correlate	Replace a dynamic value in the script with a correlation parameter.
Add as Rule	Add dynamic value definition as a rule. **Rule name.** Enables you to specify a rule name. **Application Name. Enables you to associate the rule to a specific application.** For details, see "Correlations > Rules" on page 276.

UI Element	Description
Undo Correlation	Replace the correlation parameter with the original dynamic value.
Discard	Delete the selected dynamic values from the correlation grid. You can only use the discard action when the dynamic value has a status of **New**. In addition, this action adds the text to the list of excluded strings. You can edit the list in **Recording Options > Correlation > Configuration > Excluded string list**.
View	Enables you to filter values found for correlation by the following types: • All • New • Correlated
Correlation Grid Displays details about each dynamic value in the script	
Type	Displays which engine found the dynamic value for correlations: • Record • Rules • Replay • Manual
Found/Replace	Displays information about the number of dynamic values found with the same definitions. Since you can perform partial correlation, meaning you can replace specific occurrences, the information displayed depends on if you have correlated the value or not. • **Before you correlate** Number of values that can be replaced/ Number of values found • **After you correlate** Number of values that have been replaced/Number of values found.
Status	Displays correlation status of the dynamic value from the script: • New • Correlated
Text in Response	Displays the string of the dynamic value from the script.
Correlation Parameter	Displays the correlation parameter name of the dynamic value.

UI Element	Description
Correlation Details Chevron Displays details about the dynamic value in the snapshot/script	
Original Snapshot Step Tab	
Step in Script Details	
Name	Displays the step name in the script where the dynamic value was found.
Line	Displays the line of the script where the dynamic value was found.
Action Name	Displays the name of action from the script where the dynamic value was found.
Description	Displays a description of the step.
Correlation Definition Details	
Type	Display API function that will be used to correlate the value. Regular expression: web_reg_save_param_regexp Boundary based: web_reg_save_param_ex
Definition	Displays the definition of the dynamic value. • **Regular Expression.** Dynamic value correlation is defined by a regular expression. A regular expression is a special text string for describing a search pattern. • **Boundary based.** Dynamic value correlation is defined by left and right boundary text strings.
Apply Definition	Enables you to select which definition to apply to the dynamic value. You can scroll through the definition of the dynamic value in **Occurrences in Snapshot** by clicking **Prev** or **Next** buttons.

UI Element	Description
Occurrences in Snapshot	**Record snapshot.** Displays all the occurrences of the dynamic value in the record snapshot once the script as been replayed. You can scroll to view each occurrence in the snapshot.  **Replay snapshot.**If the scan type of **Replay** has been selected, **Design Studio** displays all the occurrences of the dynamic value in the replay snapshot once the script as been replayed. You can scroll to view each occurrence in the snapshot. **Note:** Once the value has been correlated, the replay snapshot will be blank. If you modify the **Correlation Definition**, the replay snapshot will be blank.
Correlation Occurrences Tab	
Occurrences in Script	Displays the occurrences of the dynamic value in your script. You can correlate all the values or select individual values to correlate by selecting the check box adjacent to the occurrence.
Options	Opens the **Recording Options** dialog box. For details, see: "Correlations > Configuration" on page 273 "Correlations > Rules" on page 276

Async Studio

Synchronous and Asynchronous Concepts

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

Original web applications communicated using conversations that had a synchronous nature. A typical synchronous conversation includes the following steps:

1. The user interacts with an application that is presented in a web browser.
2. Based on the user input, the browser submits a request to the web server.
3. The server sends a response to the request, and the application in the browser is updated.

Synchronous applications have a number of limitations. One limitation involves the updating of the data that is displayed in the application inside the browser. For example, consider an application that displays stock prices of a number of shares. Ideally, the application should be able to update the display of the stock prices as soon as the prices are updated on the web-server. A synchronous application would be able to update the prices on a fixed time interval. For example, every 10 seconds, the browser could sent a request to the server for the most up-to-date stock prices. One limitation of this solution is that the displayed stock prices may be out-of-date for a period of time before the refresh interval is reached. Although this may not be critical in our share portfolio scenario, the scenario illustrates the limitation of a synchronous application to timeously update information.

Where necessary, synchronous applications are being replaced with what are known as *asynchronous* applications. Asynchronous applications enable a client to be notified whenever an event occurs on the server side. Asynchronous applications are therefore better able to update information as required.

In order to enable asynchronous behavior, asynchronous communication occurs in parallel (simultaneously) with the main, synchronous flow of the business processes. This behavior makes asynchronous applications harder to accurately emulate using traditional synchronous Vuser scripts.

Although there are numerous types of asynchronous applications, there are three primary types: *push*, *poll*, and *long-poll*. For details, see "Types of Asynchronous Communication" below.

For an introduction to using asynchronous communication in Vuser scripts, see "LoadRunner Support for Asynchronous Communication - Overview" on page 175.

Types of Asynchronous Communication

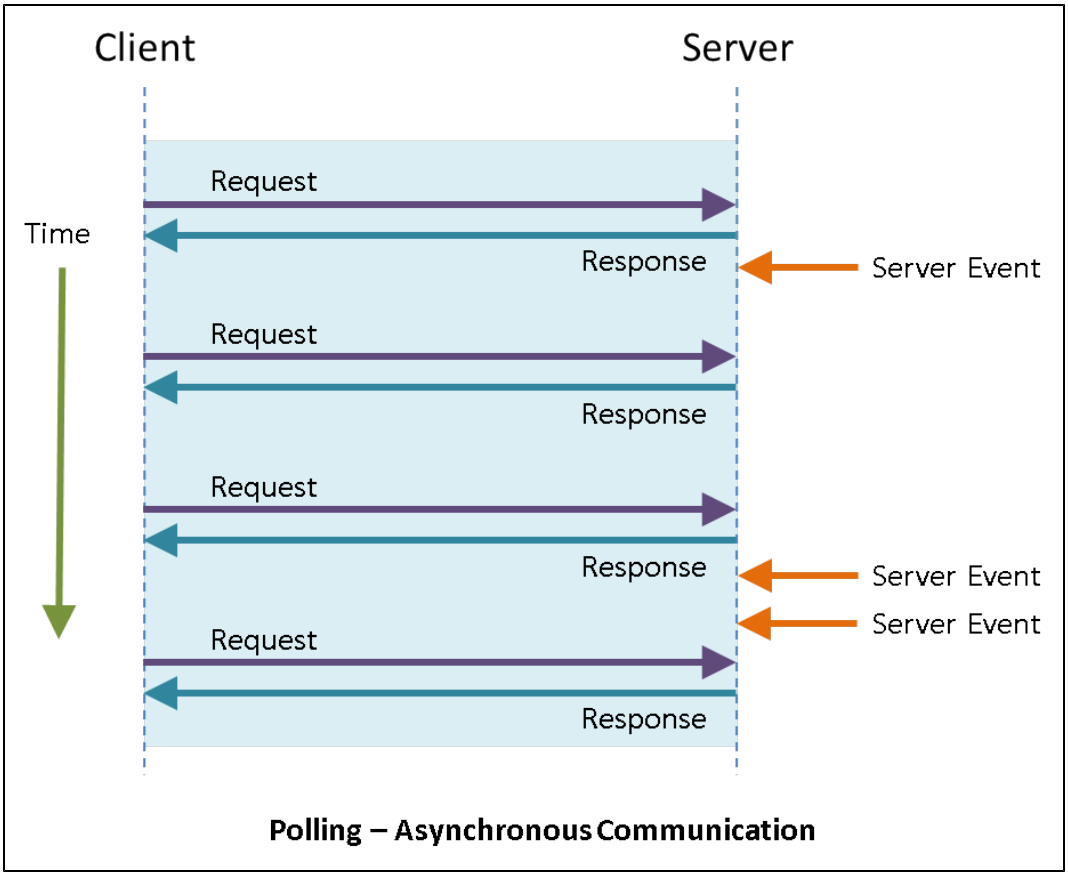
This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

Asynchronous request and response sequences

Asynchronous communication is comprised of various request and response sequences. Such request and response sequences can be classified as one of three types of asynchronous communication: *poll*, *push*, and *long-poll*. When you develop a Vuser script, it is often useful to know the types of asynchronous communication that are implemented when the required business processes are performed.

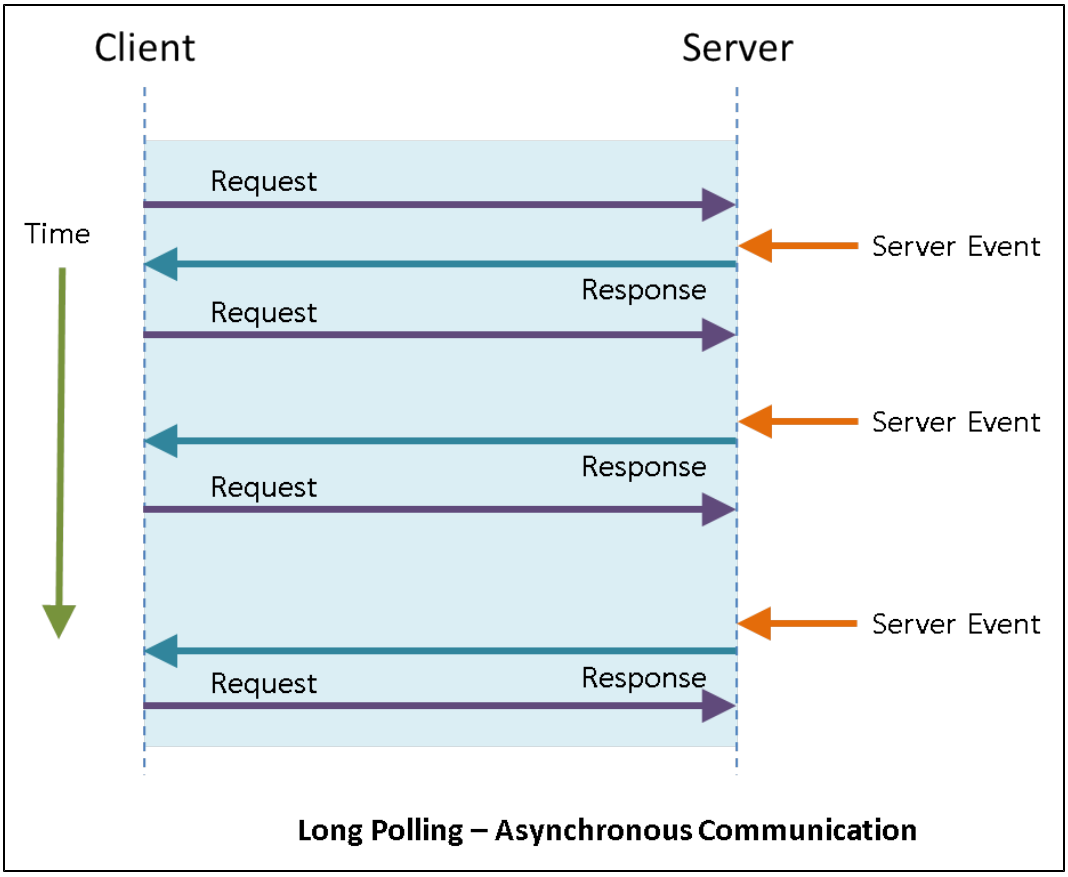
Polling Asynchronous Communication

The browser sends HTTP requests to the server at regular intervals, for example, every 5 seconds. The server responds with updates. This enables the system to intermittently update the application interface inside the browser. If the server has no update, it informs the application that there is no update, based on the application protocol.



Long-Polling Asynchronous Communication

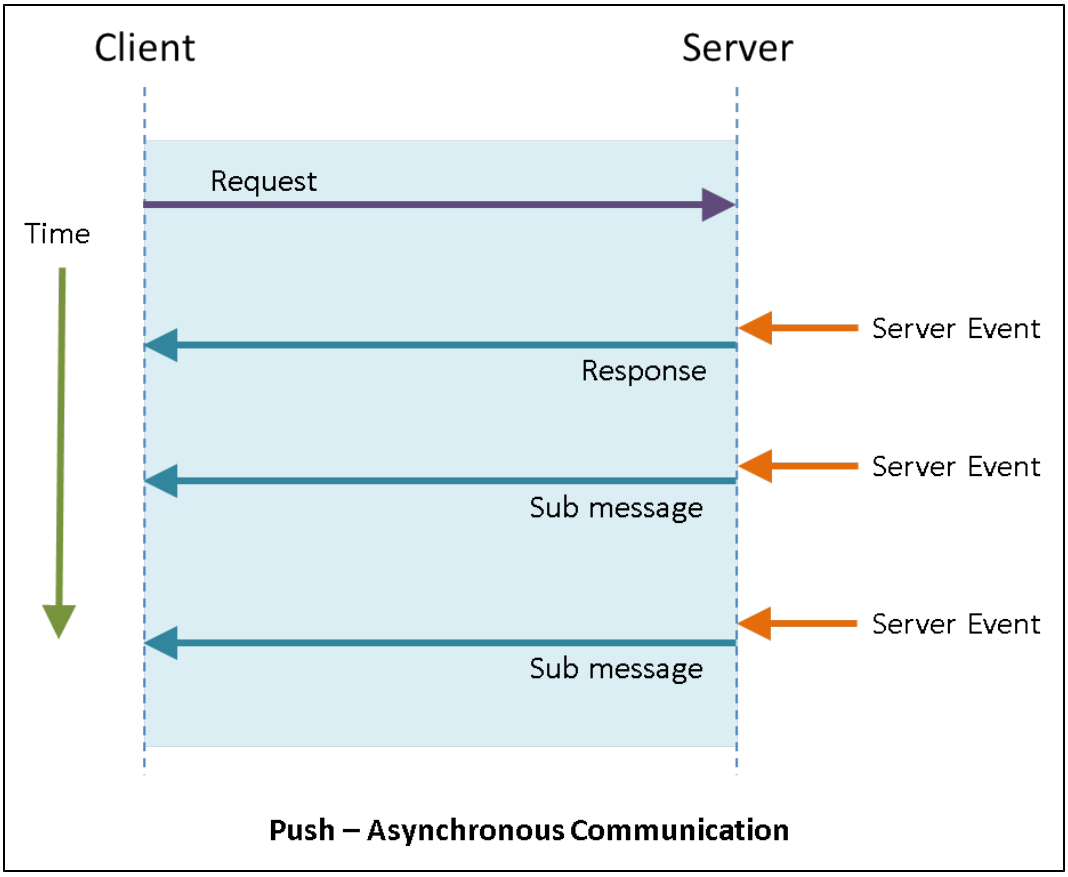
The client generates an HTTP request to a known address on the server. Whenever the server has an update, it responds with an HTTP response. Immediately after receiving the server response, the client issues another request.



Push Asynchronous Communication

The client opens a connection with the server by sending a single HTTP request to a known address on the server. The server then sends a response that appears to never end, so that the client never closes the connection. Whenever necessary, the server sends a “sub message” update to the client over the open connection. The server may or may not terminate this connection. During the time the connection is open, if the server has no real update to send, it sends "ping" messages to the client to prevent the client from closing the connection for timeout reasons.

Note that push-type conversations are supported for Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, NOT for **Flex_ amf_call** steps in Flex Vuser scripts.



LoadRunner Support for Asynchronous Communication - Overview

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

Web-based applications can exhibit synchronous behavior, asynchronous behavior, or a combination of both. For an introduction to these behavior types, see "Synchronous and Asynchronous Concepts" on page 171. LoadRunner enables you to build and run Vuser scripts that emulate user activity for both synchronous and asynchronous applications. To build a Vuser script for a synchronous application, follow the typical LoadRunner Vuser script building process. However, to build a script for an asynchronous application, you must perform some additional tasks - beyond the typical LoadRunner Vuser script building process. If you record and generate a script for an application that performs asynchronous behavior - without performing the additional asynchronous-related tasks, the script may not run successfully.

Building a Vuser script for an asynchronous application begins with recording the business processes that produce the asynchronous communication. After the business processes are recorded and the Vuser script is generated, VuGen scans the generated Vuser script and attempts to locate the asynchronous communication. If asynchronous communication is detected, VuGen modifies the script - inserting the appropriate asynchronous API functions.

Identifying Asynchronous Conversations

In order for VuGen to be able to successfully identify the asynchronous behavior in a Vuser script, the asynchronous communication must contain at least the required minimum number of request and response sequences.

- Identifying a poll-type conversation

  To enable VuGen to successfully identify a poll conversation, the recorded Vuser script must contain at least 3 sequences with matching URLs and similar polling intervals.

- Identifying a long-poll type conversation

  To enable VuGen to successfully identify a long-poll conversation, the recorded Vuser script must contain at least 3 sequences with matching URLs.

Note: VuGen will scan a script for asynchronous communication only if the **Async Scan** recording option is selected. For details, see "How to Create an Asynchronous Vuser Script" below.

In some scenarios, the modifications that VuGen makes to the Vuser script are sufficient to enable the script to run and emulate the required asynchronous behavior. In other scenarios, additional "manual" modifications are required. For details, see "How VuGen Modifies a Vuser Script for Asynchronous Communication" on page 180.

Note: the modifications that must be made to a generated Vuser script to enable the script to emulate asynchronous behavior are dependent on the type of the asynchronous behavior: *push*, *polling*, or *long-polling*. For details, see "Types of Asynchronous Communication" on page 172.

Asynchronous communication in a Vuser script is divided into one or more conversations. The individual asynchronous conversations that VuGen detects in a Vuser script are listed in the **Async** tab of the **Design Studio**. Use this list of asynchronous conversations to systematically analyze the modifications that VuGen made to the Vuser script to make sure that VuGen has correctly identified the asynchronous behavior, and correctly modified the Vuser script to emulate the required asynchronous behavior. For details on the **Async** tab of the **Design Studio**, see "Async Tab [Design Studio]" on page 193.

After modifying a Vuser script to enable it to emulate asynchronous communication, it may be necessary to perform correlation activities on the modified script. For details about correlation, see "Correlating Asynchronous Vuser Scripts" on page 186.

For details on how to build a Vuser script for an application that utilizes asynchronous communication, see "How to Create an Asynchronous Vuser Script" below.

How to Create an Asynchronous Vuser Script

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

To build a Vuser script for an asynchronous application, perform the following:

Create a new Vuser script

1. Click the **New Script** button on the VuGen toolbar.
2. Select **Web - HTTP/HTML**, or one of the other Vuser protocols that support asynchronous communication.
3. Click **Create**. VuGen creates a basic Vuser script.

Enable Async Scan

1. Select **Record > Recording Options**.
2. Under **General**, select **Code Generation**.
3. Make sure that the **Async Scan** check box is selected. This instructs VuGen to scan the Vuser script after recording, locate asynchronous communication, and insert the appropriate asynchronous functionality.

Record the business processes using the typical VuGen recording process

1. Click **Record** on the VuGen toolbar.
2. Enter the required information in the Start Recording dialog box, and then click **Start Recording**.
3. Perform the business processes that the Vuser will emulate, and then click **Stop Recording** on the floating toolbar.

> Note that in order for VuGen to be able to successfully identify the asynchronous behavior in a Vuser script, the asynchronous communication must contain at least the required minimum number of client requests and server responses. For details, see "Types of Asynchronous Communication" on page 172.

Generate, scan, and modify the Vuser script

1. After you click **Stop Recording**, VuGen generates the Vuser script.
2. After generating the script, VuGen scans the generated script to locate instances of asynchronous communication.
3. If VuGen locates any instances of asynchronous communication, VuGen will modify the script to enable the script to run and emulate the asynchronous behavior. For details, see "How VuGen Modifies a Vuser Script for Asynchronous Communication" on page 180.
4. The **Design Studio** opens. Click the **Async** tab. The **Async** tab displays a list of all instances of asynchronous communication that VuGen located in the Vuser script.

Review the modifications that VuGen made to the script

For each asynchronous conversation that appears in the **Async** tab of the **Design Studio**, perform the following tasks:

1. Open the Vuser script in the Editor.
2. Locate the **web_reg_async_attributes** step that starts the asynchronous conversation. Ensure that the **web_reg_async_attributes** step is located at the start of the asynchronous conversation.

3. Make sure that the URL parameter in the **web_reg_async_attributes** step is the same as one of the URLs that are specified in the action step that follows the **web_reg_async_attributes** step.

   For details on the **web_reg_async_attributes** step, see "Defining the Start of an Asynchronous Conversation" on page 182.

4. Notice that the step comment before the **web_reg_async_attributes** step contains a TODO token. The TODO token indicates that you should check the relevant callback implementations in the AsyncCallbacks.c extra file.

5. Locate the **web_stop_async** step that ends the asynchronous conversation. Ensure that the **web_stop_async** step is located at the end of the asynchronous conversation.

6. Make sure that the **web_stop_async** step runs as required. For details, see "Fine-Tuning the End of an Asynchronous Conversation" on page 185.

   For details on the **web_stop_async** step, see "Defining the End of an Asynchronous Conversation" on page 183.

7. Review the callback implementation and make modifications to the script as required. For details, see "Implementing Callbacks" on page 186.

8. Make sure that all *counter* and *complex string* parameters are set correctly. Notice that for each such parameter, a TODO comment exists and has a matching task in the **Tasks** pane. For details, see "Parsing URLs" on page 190.

9. Check the **Tasks** pane for specific actions that are required in order to complete the script development process. Such actions may include verifying callback implementations, or verifying the implementation of specific parameters.

10. Once all parameters are initialized correctly, run the script to make sure that the asynchronous conversation runs as expected.

Once you have reviewed the modified script and made sure that the asynchronous communication has been implemented correctly, run the script. For details, see Enabling Asynchronous Scripts to Run.

Asynchronous Communication API

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

The LoadRunner API includes several functions that enable Vuser scripts to emulate asynchronous communication. These asynchronous communication functions are:

web_reg_async_attributes

This function registers the next action function as the beginning of an asynchronous conversation, and defines the behavior of the asynchronous communication.

web_stop_async

This function cancels the specified asynchronous conversation, including all its active and future tasks.

web_sync

This function suspends the Vuser script execution until the specified parameter is defined.

web_util_set_request_url

This function sets the specified string to be the request URL for the next request sent in the conversation. This is applicable only when called from a callback.

web_util_set_request_body

This function sets the specified string to be the request body for the next request sent in the conversation. This is applicable only when called from a callback.

web_util_set_formatted_request_body

This function is similar to the **web_util_set_request_body** function. However, this function is included as part of a Flex protocol asynchronous conversation instead of a Web(HTTP/HTML) protocol asynchronous conversation. This function expects an XML formatted request body, which will be converted before the request is sent.

For details on the asynchronous API functions, see the Function Reference (**Help > Function Reference**).

The **web_reg_async_attributes** function should be called before the step that starts the asynchronous conversation. The **web_reg_async_attributes** function receives a number of arguments that define the asynchronous conversation. One of these arguments is the URL of the asynchronous conversation. As soon as the replay engine downloads this URL in the step that follows the **web_reg_async_attributes** function, the asynchronous conversation begins. The callbacks that are registered in the **web_reg_async_attributes** function enable the script developer to control some of the characteristics of the asynchronous conversation (for example, to change the URL). The asynchronous conversation continues until the **web_stop_async** step, or until the end of the iteration. In a push-type conversation, the server may close the connection and thereby end the conversation.

For details on how the asynchronous functions differ from synchronous functions, see "How Asynchronous Functions Differ from Synchronous Functions" below.

How Asynchronous Functions Differ from Synchronous Functions

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

The LoadRunner API includes several functions that enable Vuser scripts to emulate asynchronous communication. These asynchronous functions differ from the other API functions in the following ways:

1. The network traffic that the asynchronous functions generate runs in parallel – simultaneously – with the main flow in the Vuser script. This means that the asynchronous communication can continue even when the synchronous steps end.
2. The asynchronous communication continues even during execution of non-web functions (e.g. **lr_think_time**).

3. Some of the asynchronous communication API functions use callback functions. The user needs to specify callbacks that are scheduled by the replay engine when a predefined event occurs. For details on using callbacks with asynchronous functions, see "Implementing Callbacks" on page 186.

How VuGen Modifies a Vuser Script for Asynchronous Communication

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After you create a Vuser script and record the required business processes, VuGen generates the Vuser script. VuGen then scans the generated script to locate instances of asynchronous communication. This process is called an Async scan. If VuGen locates any instances of asynchronous communication in the Vuser script, VuGen modifies the script to enable the script to run and emulate the required asynchronous behavior.

Note: VuGen will scan a script for asynchronous communication only if the **Async Scan** recording option is selected. For details, see "How to Create an Asynchronous Vuser Script" on page 176.

Asynchronous communication in a Vuser script is divided into one or more conversations. The individual asynchronous conversations that VuGen detects in a Vuser script are listed in the **Async** tab of the **Design Studio**. Use this list of asynchronous conversations to systematically analyze the modifications that VuGen made to the Vuser script during the Async scan. Make sure that VuGen has correctly identified the asynchronous behavior in the Vuser script, and correctly modified the Vuser script to emulate the required asynchronous behavior. For details on the **Async** tab of the **Design Studio**, see "Async Tab [Design Studio]" on page 193.

Note: After modifying a Vuser script to enable it to emulate asynchronous communication, it may be necessary to perform correlation activities on the modified script. For details about correlation, see "Correlating Asynchronous Vuser Scripts" on page 186.

How does VuGen modify a Vuser script?

Asynchronous behavior in a Vuser script is divided into one or more asynchronous conversations. For each asynchronous conversation, VuGen performs the following tasks:

1. VuGen inserts a **web_reg_async_attributes** step before the start of the asynchronous conversation. The **web_reg_async_attributes** step includes an ID for the asynchronous conversation. This ID is used by a subsequent **web_stop_async** step to indicate the end of the asynchronous conversation. The Pattern argument indicates the type of the asynchronous behavior: *push*, *poll*, or *long-poll*.

```
web_reg_async_attributes("ID=Push_0",
    "URL=http://push.example.com/example",
    "Pattern=Push",
    "RequestCB=Push_0_RequestCB",
    "ResponseBodyBufferCB=Push_0_ResponseBodyBufferCB",
    "ResponseCB=Push_0_ResponseCB",
    LAST);
```

For details on how a **web_reg_async_attributes** step is used at the start of an asynchronous conversation, see "Defining the Start of an Asynchronous Conversation" on next page.

For details on the **web_reg_async_attributes** function, see the Function Reference (**Help > Function Reference**).

For details on the types of asynchronous behavior that are supported by LoadRunner, see "Types of Asynchronous Communication" on page 172.

2. VuGen adds a comment before the inserted **web_reg_async_attributes** step. The comment includes details about the asynchronous conversation, including:
   a. The ID of the asynchronous conversation.
   b. The URLs that are included in the conversation.
   c. Suggested implementations for the callback functions that are declared in the **web_reg_async_attributes** step. These implementations are added in AsyncCallbacks.c, one of the script's extra files.

```
/* Added by Async CodeGen.
ID=Push_0
ScanType = Recording

The following urls are considered part of this conversation:
    http://push.example.com/example

TODO - The following callbacks have been added to AsyncCallbacks.c.
Add your code to the callback implementations as necessary.
    Push_0_RequestCB
    Push_0_ResponseBodyBufferCB
    Push_0_ResponseCB
 */
```

3. For *push* conversations, VuGen inserts asynchronous API functions into the Vuser script, but does not remove any of the recorded code from the Vuser script. For *polling* and *long-polling* conversations, VuGen may remove steps or step parameters from the generated Vuser script. VuGen removes steps or step parameters in cases where the relevant URLs will be requested by running the inserted asynchronous functions - and not by running the original steps that have been removed.

   Note: Removed steps are not deleted – they are commented out. You can "uncomment" these steps if required.

4. When relevant, VuGen adds a **web_stop_async** step at the end of the asynchronous conversation. The **web_stop_async** step marks the end of the asynchronous conversation. For details on the **web_stop_async** step, see the Function Reference (**Help > Function Reference**).

5. The recording snapshots are updated by grouping the tasks in the asynchronous conversation under the step that started the conversation.

How VuGen modifies flex_amf_call steps

VuGen supports asynchronous polling and long-polling behavior in **flex_amf_call** steps. Flex scripts that contain *polling* or *long-polling* in **flex_amf_call** steps are handled by VuGen just like Web(HTTP/HTML) scripts, except for the following:

- The RequestCB will contain a commented call to **web_util_set_formatted_request_body**, which can be used to pass an XML formatted request body, which will be encoded and sent with the request.
- The **aResponseBodyStr** and **aResponseBodyLen** parameters of the ResponseCB give user access to the XML representation of the response body.

Defining the Start of an Asynchronous Conversation

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After VuGen scans a Vuser script for asynchronous communication, the Async tab of the Design Studio lists the asynchronous conversations that VuGen found in the script. VuGen inserts a **web_reg_async_attributes** into the Vuser script at the start of each asynchronous conversation that was detected. Use VuGen's Step Navigator to find the associated **web_reg_async_attributes** steps in the Vuser script. The **web_reg_async_attributes** steps should be located where the asynchronous conversations start when the script runs.

A **web_reg_async_attributes** step that is added to a Vuser script includes the following parameters:

- ID
- URL
- Pattern
- PollIntervalMS (for poll-type conversations only)
- RequestCB
- ResponseBodyBufferCB (for push-type conversations only)
- ResponseCB

The URL parameter in the **web_reg_async_attributes** step should be the same as one of the URLs that are specified in the step that follows the **web_reg_async_attributes** step. For details on the **web_reg_async_attributes** step, see the Function Reference (**Help > Function Reference**).

Inserting a Comment

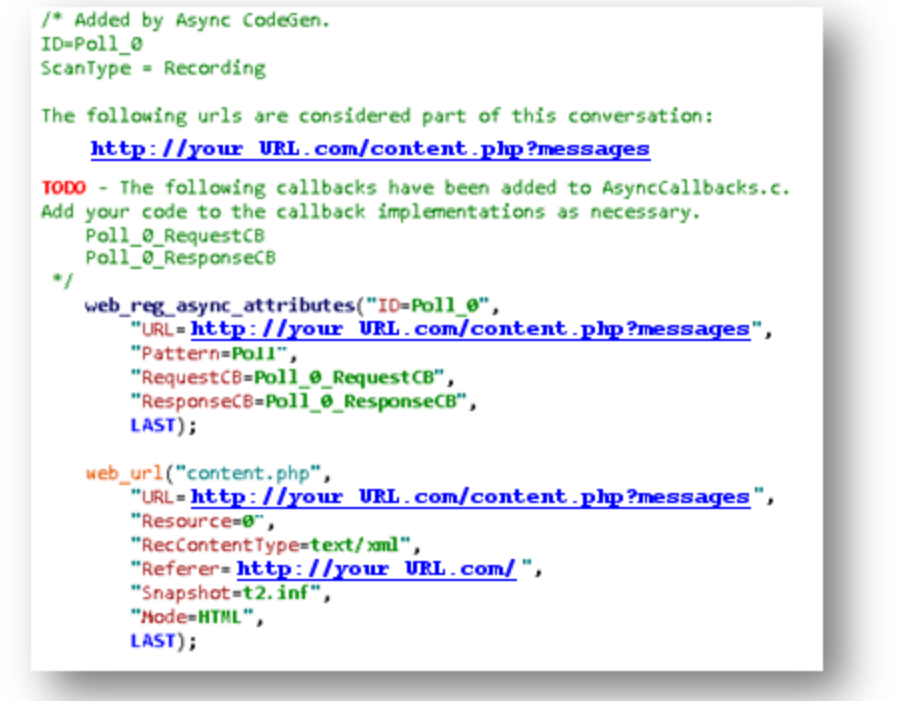
When VuGen inserts a **web_reg_async_attributes** step into a script, VuGen inserts an associated comment before the **web_reg_async_attributes** step. The inserted comment contains information about the associated asynchronous conversation, such as the conversation ID, the communication pattern (*push*, *poll*, or *long-poll*), a list of URLs that are part of the asynchronous communication, and list of callbacks implemented in the AsyncCallbacks.c extra file.

Notice that the step comment contains a TODO token. The TODO token indicates that you should check the relevant callback implementations in the AsyncCallbacks.c extra file.

For details on how an asynchronous conversation is terminated, see "Defining the End of an Asynchronous Conversation" on next page.

Example - web_reg_async_attributes

The sample code below shows a **web_reg_async_attributes** step that was added by VuGen. Notice that the **web_reg_async_attributes** step was added before a **web_url** step, and that the URL parameter in the **web_reg_async_attributes** step is the same as the URL parameter in the **web_url** step.



```
/* Added by Async CodeGen.
ID=Poll_0
ScanType = Recording

The following urls are considered part of this conversation:
    http://your_URL.com/content.php?messages
TODO - The following callbacks have been added to AsyncCallbacks.c.
Add your code to the callback implementations as necessary.
    Poll_0_RequestCB
    Poll_0_ResponseCB
 */
    web_reg_async_attributes("ID=Poll_0",
        "URL=http://your_URL.com/content.php?messages",
        "Pattern=Poll",
        "RequestCB=Poll_0_RequestCB",
        "ResponseCB=Poll_0_ResponseCB",
        LAST);

    web_url("content.php",
        "URL=http://your_URL.com/content.php?messages",
        "Resource=0",
        "RecContentType=text/xml",
        "Referer=http://your_URL.com/",
        "Snapshot=t2.inf",
        "Mode=HTML",
        LAST);
```

Defining the End of an Asynchronous Conversation

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After VuGen scans a Vuser script for asynchronous communication, the Async tab of the Design Studio lists the asynchronous conversations that VuGen found in the script. A **web_stop_async** step is inserted into the Vuser script at the end of each asynchronous conversation that was detected. Use VuGen's Step Navigator to find the associated **web_stop_async** steps in the Vuser script.

Note: In some cases VuGen will not add a **web_stop_async** step at the end of an asynchronous conversation. This may occur when VuGen is not able to determine where the asynchronous conversation ends. This can occur when the asynchronous conversation was added due to a specific Async rule or when the asynchronous conversation was not ended during the recording. For details on Async rules, see "Async Rules" on page 192.

After VuGen has inserted a **web_stop_async** step into a Vuser script, make sure the **web_stop_async** step was added in the correct location in the script, that is – where the asynchronous conversation should end when the Vuser script runs.

In order to make sure the asynchronous conversation ends correctly when the script runs, it may be necessary to modify the details of the **web_stop_async** step in the Vuser script. For details, see "Fine-Tuning the End of an Asynchronous Conversation" on next page.

For details on how an asynchronous conversation is started, see "Defining the Start of an Asynchronous Conversation" on page 182.

Example - web_stop_async:

In the code sample below, VuGen added a **web_stop_async** step at the end of a *poll* conversation. In this example, the original polling steps are commented out, and the **lr_think_time** steps that separated them have been merged into one **lr_think_time** step in order to emulate the duration of the entire *poll* conversation.

```
/* Removed by Async CodeGen.
ID = Poll_0
*/
    /*
web_url("content.php_7",
        "URL=http://your URL.com/content.php?messages",
        "Resource=0",
        "RecContentType=text/xml",
        "Referer=http://your URL.com/",
        "Snapshot=t8.inf",
        "Mode=HTML",
        LAST);
    */


    lr_think_time(24);


    web_stop_async("ID=Poll_0",
        LAST);
```

Using Asynchronous Request Thresholds

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

You can fine-tune some of VuGen's behavior when VuGen scans a Vuser script to locate asynchronous communication. You use VuGen's asynchronous request thresholds to fine-tune VuGen's behavior. Each of the thresholds is associated with only one of the types of asynchronous conversations: *push*, *poll*, or *long-poll*.

- **Asynchronous request thresholds for push conversations**

  Minimum Response Size. Specify the minimum response content length (in bytes) for defining *push* asynchronous conversations. If the server sent less than the specified value, VuGen will not classify the conversation as a push-type asynchronous conversation.

  Maximum Sub Message Size. Specify the maximum sub message size (in bytes) sent by the server for defining *push* asynchronous conversations. If the server sent a sub message of size

greater than the specified value, VuGen will not classify the conversation as a push-type asynchronous conversation.

Minimum Number of Sub Messages. Specify the minimum number of sub messages for defining *push* asynchronous conversations. A push conversation in which less than the specified number of sub messages was sent by the server will not be classified by VuGen as a push-type asynchronous conversation.

- **Asynchronous request thresholds for poll conversations**

  Interval Tolerance. Specify the interval tolerance (in milliseconds) for classifying *poll* asynchronous conversations. A conversation in which intervals differ from each other by more than the specified value will not be classified by VuGen as a poll-type asynchronous conversation.

- **Asynchronous request thresholds for long-poll conversations**

  Maximum Interval. Specify the maximum interval (in milliseconds) between the end of one response and the start of a new request for classifying *long-poll* asynchronous conversations. A conversation in which a request starts more than the specified value after the end of the previous response will not be classified by VuGen as a long-poll type asynchronous conversation.

For details on the available asynchronous request thresholds, see "Asynchronous Request Thresholds Dialog Box" on page 194.

Fine-Tuning the End of an Asynchronous Conversation

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After VuGen scans a Vuser script for asynchronous communication, the Async tab of the Design Studio lists the asynchronous conversations that were found in the script. A **web_stop_async** step is inserted into the Vuser script at the end of each asynchronous conversation that was detected. In order to make sure that each asynchronous conversation ends correctly when the script runs, it may be necessary to perform one or more of the following tasks:

- Remove the **web_stop_async** step so that the asynchronous conversation will be terminated at the end of the iteration.
- Move the **web_stop_async** step to a location that is after an existing action step or an existing **lr_think_time** step, so the asynchronous conversation will end after that step is performed.
- Add an **lr_think_time** step before the **web_stop_async** step, or change the time parameter in an existing **lr_think_time** step. Make sure that think-time is enabled in the Run-Time Settings. For details, see "General > Think Time Node" on page 361.
- Add a **web_sync** step to stop the asynchronous conversation after a specified parameter receives a value. Use the asynchronous conversation callbacks to make sure the parameter receives a value only when you want to end the conversation.

Correlating Asynchronous Vuser Scripts

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After modifying a Vuser script to enable it to emulate asynchronous communication, it may be necessary to perform correlation activities on the modified script. Due to asynchronous nature, dynamic values from asynchronous communication cannot be handled by Design Studio, and must be correlated manually.

You can search for dynamic values inside Response callbacks functions using the **lr_save_param_regexp** function. This function can be called from a callback to extract the necessary value from server response (**ResponseCB**) or response buffer (**ResponseBodyBufferCB**), and assign this value to a parameter. This parameter can then be used for correlations.

For details about the **lr_save_param_regexp** function, see the Function Reference (**Help > Function Reference**).

Implementing Callbacks

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After VuGen scans a Vuser script for asynchronous communication, the **Async** tab of the **Design Studio** lists the asynchronous conversations that were found in the script. For each asynchronous conversation found during the scan, VuGen adds the callback function signatures matching those declared in the **web_reg_async_attributes** step. The signatures are added to the AsyncCallbacks.c extra file.

The available callbacks are:

1. RequestCB

   This callback is called before a request is sent.

2. ResponseBodyBufferCB

   This callback is called when there is content in the response body buffer and at the end of the response body. This callback is generated by VuGen automatically for *push*-type conversations, but is available for *poll* and *long-pole* conversations as well.

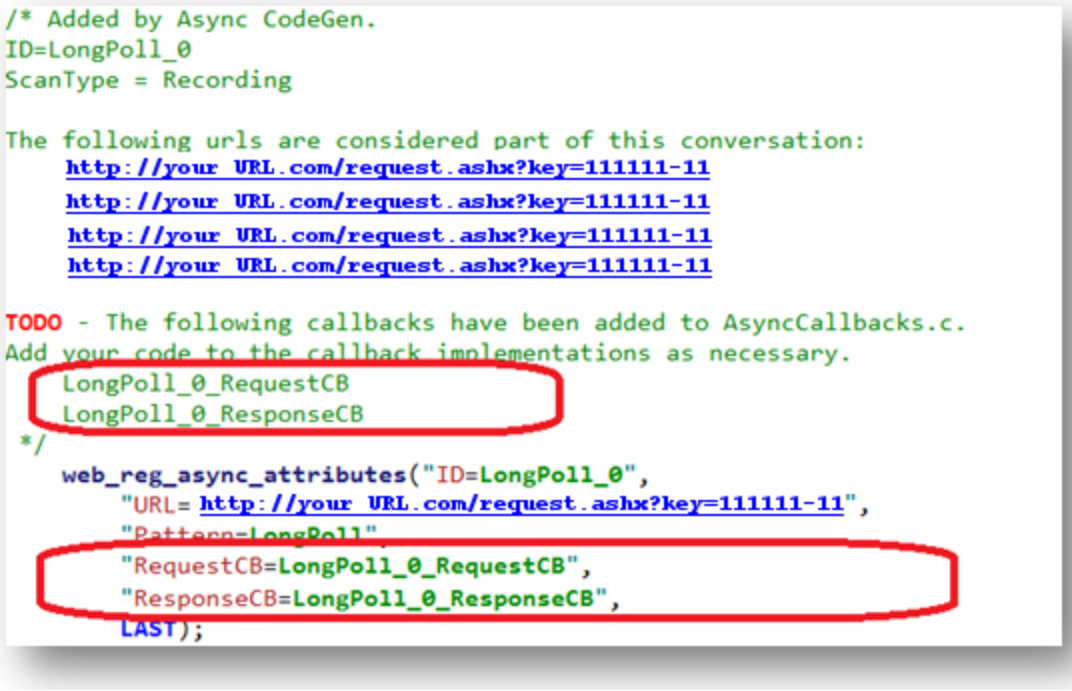
3. ResponseCB

   This callback is called after every response is received in the conversation.

- The names of the callback functions start with the conversation ID of the asynchronous conversation. For example, the RequestCB callback for an asynchronous conversation with ID “LongPoll_0” will be **LongPoll_0_RequestCB**.
- The names of the callback functions are declared in the **web_reg_async_attributes** step in the script.

Example 1:

In the following sample code, the three callback functions are declared in the **web_reg_async_attributes** step.



Example 2:

In the following sample code, the two callbacks are implemented in the AsyncCallbacks.c extra file.

```
int LongPoll_0_RequestCB()
{
    //enter your implementation for RequestCB() here

    //call web_util_set_request_url() here to modify polling url
    //web_util_set_request_url("<request_url>");

    //call web_util_set_request_body() here to modify request body:
    //web_util_set_request_body("<request body>");

    return WEB_ASYNC_CB_RC_OK;
}

int LongPoll_0_ResponseCB(
    const char *    aResponseHeadersStr,
    int             aResponseHeadersLen,
    const char *    aResponseBodyStr,
    int             aResponseBodyLen,
    int             aHttpStatusCode)
{
    //enter your implementation for ResponseCB() here

    return WEB_ASYNC_CB_RC_OK;
}
```

You can modify the callbacks to implement the required behavior. For details, see "Modifying Callbacks" below.

Modifying Callbacks

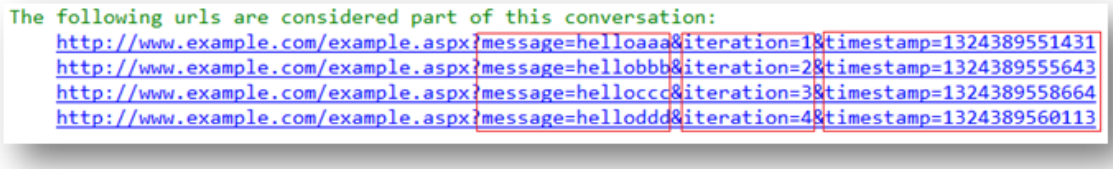
This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

After VuGen scans a Vuser script for asynchronous communication, the **Async** tab of the **Design Studio** lists the asynchronous conversations that were found in the script. For each asynchronous conversation found during the scan, VuGen adds the required callback function declarations in the AsyncCallbacks.c extra file. To implement the required behavior, you can modify the callbacks that were added by VuGen. Modifying a callback includes:

Modifying the request URL in the RequestCB callback

In *poll* and *long-poll* conversations, requested URLs often change in each polling iteration. The change is usually determined by client-side logic, usually implemented by JavaScript that is executed by the browser. Parts of the URL may be determined by correlation of a known parameter, such as a session ID. For details, see "Parsing URLs" on page 190.

Request URLs in an asynchronous conversation will be modified before the request is sent by using the RequestCB and the **web_util_set_request_url** function.



Modifying the request body in the RequestCB callback

The request body in requests that are part of an asynchronous conversation may need to be modified before the request is sent. You use the RequestCB and the **web_util_set_request_body** util function to modify the request body.

Modifying the request body is useful in *poll* and *long-poll* conversations in which each new request requires a different request body.

```
//an example of a parametrized request body sent in the RequestCB.
//the value of {request_body} may be set in the callback function,
//or elsewhere in the script.
web_util_set_request_body("{request_body}");
```

Each RequestCB that is generated by VuGen contains a commented snippet. You can "uncomment" the snippet in order to use the **web_util_set_request_body** util function.

If VuGen finds that different requests have different body values in the recorded conversation, the generated RequestCB will contain a comment that prompts you to check the recording in order to implement the request body sent in each request when the script runs.

```
//call web_util_set_request_body() here to modify request body:
//web_util_set_request_body("<request body>");
//TODO - use snapshot view to see examples of request bodies sent
```

Modifying callbacks in Flex Vuser scripts

For Flex asynchronous *polling* and *long-polling* conversations, the generated RequestCB contains a comment that prompts you to uncomment the calls to **lr_save_string** and to **web_util_set_formatted_request_body**, in order to set **an XML formatted request body** for each request.

Correlations applied to the original request body in the Action file must be manually applied to the formatted request body.

```
int Poll_0_RequestCB()
{
    //enter your implementation for RequestCB() here

    //call web_util_set_request_url() here to modify request url:
    //web_util_set_request_url("<request url>");

    //TODO - apply correlations from the matching flex_amf_call step to the following formatted body.
    //use snapshot view to see examples of request bodies sent.
    //uncomment the following section to modify the formatted request body.
    lr_save_string(
        lr_eval_string(
        "<AMFPacket AMF_version=\"3\">"
      "<AMFHeaders />"
      "<Messages>"
        "<Message method=\"null\" target=\"/6\">"

        ...

        "</Message>"
      "</Messages>"
    "</AMFPacket>"),
    "Poll_0_formatted_body_param");

    //TODO - call web_util_set_formatted_request_body() here to set the request body.
    //notice that web_util_set_formatted_request_body() expectes an xml formatted request body.
    web_util_set_formatted_request_body("{Poll_0_formatted_body_param}");

    return WEB_ASYNC_CB_RC_OK;
}
```

For details on using callback functions, see "Implementing Callbacks" on page 186.

Parsing URLs

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

URLs that are included in asynchronous conversations often include query strings that are derived in a variety of ways. These strings may include:

- Time-stamps
- Counters
- Complex strings

To enable a Vuser script to successfully perform asynchronous communication, VuGen must be able to recreate the required URLs.

- When the URL includes a time-stamp, VuGen is usually able to successfully create the required URL.
- When the URL includes a counter, VuGen is usually able to recreate the counter, but it may be necessary to manually initialize the counter in the script.
- When the URL includes more complex strings, the algorithms for generating the URLs must be manually added to the code in the Vuser script.

Example:

The sample code below shows a set of URLs that are part of a *long-poll* conversation. The URLs are included in the comment generated for a **web_reg_async_attributes** step:

```
The following urls are considered part of this conversation:
    http://www.example.com/example.aspx?message=helloaaa&iteration=1&timestamp=1324389551431
    http://www.example.com/example.aspx?message=hellobbb&iteration=2&timestamp=1324389555643
    http://www.example.com/example.aspx?message=helloccc&iteration=3&timestamp=1324389558664
    http://www.example.com/example.aspx?message=hellodddd&iteration=4&timestamp=1324389560113
```

If none of the parameters shown in the code sample above was found in VuGen's scan of the recorded Vuser script, the RequestCB implementation will contain a snippet that may be uncommented in order to set the URL for each response according to user defined code. For details, see "Modifying Callbacks" on page 188.

```
//call web_util_set_request_url() here to modify request url:
//web_util_set_request_url("<request url>");
```

If any or all of the parameters shown in the sample code above are found during VuGen's scan of the recorded Vuser script, the RequestCB implementation will contain the following:

- A comment prompting the user to call **web_util_set_request_url**. The comment will contain a parameterized version of the URL.
- For each *time-stamp* parameter found in the URL, a snippet for saving the time-stamp to a parameter.
- For each *counter* parameter found in the URL, a snippet for incrementing a counter parameter. A matching step for initializing the counter parameter will also be added to the Action file. The snippet will also contain examples of the URL token containing the counter parameter, as seen during the recording.
- For each *complex string* parameter found in the URL, a snippet for saving the string to a parameter. It is up to the user to generate the correct string to be saved to the parameter to be used in the URL. The snippet will also contain examples of the URL token that is considered an unknown parameter, as seen during the recording.
- A snippet for passing the parameterized version of the URL to the **web_util_set_request_url** function.

Example: A snippet containing the parameterized version of a URL.

```
//call web_util_set_request_url() here to modify polling url
//url is expected to be of the form:
//http://www.example.com/example.aspx?message={Unknown_LongPoll_0_0}
//&iteration={Counter_LongPoll_0_1}&timestamp={TimeStamp_LongPoll_0_2}
```

Example: A snippet prompting the user to set the value of an unknown parameter.

```
//TODO - implement parameter of type unknown: Unknown_LongPoll_0_0.
//Known examples for Unknown_LongPoll_0_0:
//message=[{"channel":"\/meta\/connect","connectionType":"long-polling","id":3,"clientId":"113fc44"}],
//message=[{"channel":"\/meta\/connect","connectionType":"long-polling","id":5,"clientId":"113fc44"}],
//message=[{"channel":"\/meta\/connect","connectionType":"long-polling","id":6,"clientId":"113fc44"}],
//message=[{"channel":"\/meta\/connect","connectionType":"long-polling","id":7,"clientId":"113fc44"}].
lr_save_string("[{\"channel\":\"\\/meta\\/connect\",\"connectionType\":\"long-polling\",\"id\":3,\"clientId\":\"113fc44\"}]",
"Unknown_LongPoll_0_0");
```

Example: A snippet for incrementing a counter parameter.

```
//TODO - check counter initialization for Counter_LongPoll_0_1 in Action file.
//Known examples for the token containing Counter_LongPoll_0_1:
//iteration=1, iteration=2, iteration=3, iteration=4 gPoll_0_1:
lr_param_increment("Counter_LongPoll_0_1", "{Counter_LongPoll_0_1}");
```

Example: A snippet for initializing a counter parameter.

```
lr_save_int(0, "Counter_LongPoll_0_1");
```

Example: A snippet for saving a timestamp parameter.

```
web_save_timestamp_param("TimeStamp_LongPoll_0_2",LAST);
```

Example: A snippet for passing the parameterized version of a URL to the **web_util_set_request_url** function.

```
//once all parameters have been assigned, copy them to the updated url,
//and call web_util_set_request_url() with the updated url:
web_util_set_request_url("http://www.example.com/example.aspx?message={Unknown_LongPoll_0_0}
&iteration={Counter_LongPoll_0_1}&timestamp={TimeStamp_LongPoll_0_2}");
```

Async Rules

This topic applies to Web (HTTP/HTML) protocol actions inside Web (HTTP/HTML), Flex, Silverlight, and Web Services Vuser scripts, and **Flex_amf_call** steps in Flex Vuser scripts.

In some cases, when VuGen performs an Async scan, VuGen may fail to identify asynchronous conversations that are included in the Vuser script. In other cases, VuGen may erroneously classify regular synchronous steps as part of an asynchronous conversation. To help rectify both of these scenarios, you can use LoadRunner's Async rules file to define rules that determine how specified URLs and regular expressions will be classified during an Async scan.

For details about using the Async rules functionality, contact HP Software Support.

Async Tab [Design Studio]

The Async tab of the Design Studio lists all the occurrences of asynchronous communication that VuGen detected in the Vuser script.

To access	• Select **Design > Design Studio**, and then click the **Async** tab. • Click the **Design Studio** button on the VuGen toolbar, and then click the **Async** tab.
Important information	• The **Design Studio** button is enabled only when you display a recorded Vuser script in the Solution Explorer. • The Async tab enables you to only view the asynchronous communication that is included in the Vuser script - you cannot edit any of the asynchronous details from the Async tab. Changes to the asynchronous details must be made in the Vuser script.
Relevant tasks	"How to Create an Asynchronous Vuser Script" on page 176

User interface elements are described below:

UI Element	Description
Type	Indicates the origin of the asynchronous code in the Vuser script: **Record.** The asynchronous code was added by VuGen during an Async scan that was performed after recording or regenerating the Vuser script. **Rule.** The asynchronous code was added by VuGen due to a specific rule in the Async rules file. **Manual.** The asynchronous code was manually added by a user.
Action	The section of the Vuser script in which the asynchronous behavior is located.
Occurrences	• For push-type conversations, **Occurrences** is always 1. • For poll and long-poll conversations, **Occurrences** indicates the number of steps or extra resource attributes that were removed [commented-out] by VuGen during the Async scan of the Vuser script.
Status	Always has the value **Applied**.
Async Type	The type of the asynchronous behavior that was detected: **Push**, **Poll**, or **Long-Poll**.
URL	The URL in the **web_reg_async_attributes** step that starts the asynchronous conversation.
Filter	Select which asynchronous conversations to display in the conversation list.

UI Element	Description
Details	Expands the dialog box to show details about the selected asynchronous conversation.
Name	Always has the value **web_reg_async_attributes**.
Line	The line in the Vuser script that contains the **web_reg_async_attributes** step.
Action Name	The section of the Vuser script in which the asynchronous behavior is located.
Description	The comment in the Vuser script that precedes the **web_reg_async_attributes** step.
Occurrences in Snapshot	• For push-type conversations, displays the response body. • For poll and long-poll conversations, displays HTTP attributes associated with the asynchronous conversation.
Options	Opens the Asynchronous Request Thresholds dialog box.

Asynchronous Request Thresholds Dialog Box

This dialog box enables you to fine-tune some of VuGen's behavior when VuGen scans a Vuser script to locate asynchronous communication.

To access	**VuGen > Record > Recording Options > General > Code Generation** and then click **Async Options**

User interface elements are described below:

UI Element	Description
Minimum Response Size	Specify the minimum size (in bytes) of a server response for defining *push* asynchronous conversations. If the server sent less than the specified value, VuGen will not classify the conversation as a push-type asynchronous conversation.
Maximum Sub Message Size	Specify the maximum sub message size (in bytes) sent by the server for defining *push* asynchronous conversations. If the server sent a sub message of size greater than the specified value, VuGen will not classify the conversation as a push-type asynchronous conversation.
Minimum Number of Sub Messages	Specify the minimum number of valid sub messages for defining *push* asynchronous conversations. A push conversation in which less than the specified number of valid sub messages was sent by the server will not be classified by VuGen as a push-type asynchronous conversation.

Interval Tolerance	Specify the interval tolerance (in milliseconds) for classifying *poll* asynchronous conversations. A conversation in which intervals differ from each other by more than the specified value will not be classified by VuGen as a poll-type asynchronous conversation.
Maximum Interval	Specify the maximum interval (in milliseconds) between the end of one response and the start of a new request for classifying *long poll* asynchronous conversations. A conversation in which a request starts more than the specified value after the end of the previous response will not be classified by VuGen as a long-poll type asynchronous conversation.

Asynchronous Example - Poll

The following example describes a Vuser script that includes a poll asynchronous conversation. The application that is emulated by the Vuser is a demo of a "wiki" page. The browser displays the wiki page, and sends requests to refresh the page at intervals of 5 seconds.

```
Action()
{

    web_url("wiki",
        "URL=http://example.com/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);

    web_url("content.php",
        "URL=http://example.com/content.php?messages",
        "Resource=0",
        "RecContentType=text/xml",
        "Referer=http://example.com/",
        "Snapshot=t2.inf",
        "Mode=HTML",
        LAST);

    lr_think_time(4);

    web_url("content.php_2",
        "URL=http://example.com/content.php?messages",
        "Resource=0",http://example.com/content.php?messages",
        "RecContentType=text/xml",
        "Referer=http://example.com/",
        "Snapshot=t3.inf",
        "Mode=HTML",
        LAST);

    lr_think_time(4);

    web_url("content.php_3",
        "URL=http://example.com/content.php?messages",
        "Resource=0",
        "RecContentType=text/xml",
        "Referer=http://example.com/",
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);
```

Note: You can modify VuGen's asynchronous request thresholds to assist VuGen in finding poll-type conversations. For details, see "Using Asynchronous Request Thresholds" on page 184.

The above script was generated by VuGen after the required business processes were recorded. An asynchronous scan was not performed on the script after the script was generated. Notice that the script contains a series of **web_url** functions with a repeating URL, namely: **http://example.com/content.php?messages**. These **web_url** functions are separated by **lr_think_time** functions, indicating that the **web_url** functions repeat at intervals of 4 seconds.

When the Vuser script runs, requests for **http://example.com/content.php?messages** should be sent repeatedly until the script is finished. Additionally, these requests should be sent in parallel (simultaneously) with other actions performed in the Vuser script.

After VuGen performed an asynchronous scan on the script, the script looks as follows:

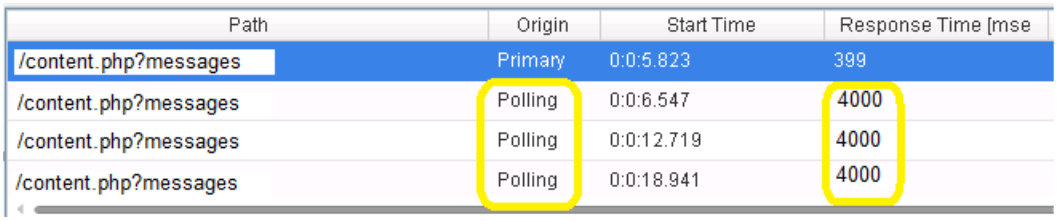
```
    /* Added by Async CodeGen.
ID=Poll_0
ScanType = Recording

The following urls are considered part of this conversation:
    http://example.com/content.php?messages

TODO - The following callbacks have been added to AsyncCallbacks.c.
Add your code to the callback implementations as necessary.
    Poll_0_RequestCB
    Poll_0_ResponseCB
 */
    web_reg_async_attributes("ID=Poll_0",
        "URL=http://example.com/content.php?messages",
        "Pattern=Poll",
        "PollIntervalMs=4000",
        "RequestCB=Poll_0_RequestCB",
        "ResponseCB=Poll_0_ResponseCB",
        LAST);


    web_url("content.php",
        "URL=http://example.com/content.php?messages",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://example.com",
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);
```

Notice that a **web_reg_async_attributes** function has been added to the script before the first **web_url** function that calls **http://example.com/content.php?messages**.

Except for the first call to **http://example.com/content.php?messages**, all other **web_url** functions that call the same URL have been commented-out by VuGen.

Notice that the **lr_think_time** functions have been merged into one **lr_think_time** function.

The Snapshot pane for the remaining **web_url** function shows that the snapshots for the removed **web_url** functions now have Origin = Polling, and that they start at intervals of 5 seconds.



Path	Origin	Start Time	Response Time [mse
/content.php?messages	Primary	0:0:5.823	399
/content.php?messages	Polling	0:0:6.547	4000
/content.php?messages	Polling	0:0:12.719	4000
/content.php?messages	Polling	0:0:18.941	4000

Since the requests also have a response time, the think time in **lr_think_time** functions between the polling steps in the original script has been rounded down to 4 seconds.

Asynchronous Example - Push

The following example describes a Vuser script that is developed to emulate a browser displaying an application that utilizes push-type asynchronous conversations. The application is a demo of a “stock quote” page. The browser shows the page with the stock values, and then sends a request and receives a response with updated stock values. The request remains open until it is closed by the user. For as long as the page is displayed, the server will continue to send sub-messages as part of the response - whenever the server has an update for the displayed stocks. Whenever such a sub-message is received by the client, the client displays the updated stock values.

Note: You can modify VuGen's asynchronous request thresholds to assist VuGen in finding push-type conversations. For details, see "Using Asynchronous Request Thresholds" on page 184.

Name	Price	Time	Change	Bid Size	Bid	Ask	Ask Size	Min	Max	Ref.
Anduct	3.09	10:32:54	1.64	5000	3.09	3.1	35500	2.48	3.64	3.04
Ations Europe	17.92	10:33:05	11.37	86000	17.92	17.98	1000	12.8	19.33	16.09
Bagies Consulting	6.43	10:33:04	-10.57	4500	6.43	6.44	8500	5.74	8.64	7.19
BAY Corporation	3.2	10:32:03	-11.84	37500	3.2	3.21	63500	3.13	4.28	3.63
CON Consulting	6.8	10:33:05	-10.64	24000	6.8	6.83	51000	6.23	9.08	7.61
Corcor PLC	2.58	10:32:54	12.17	61000	2.58	2.59	42000	1.87	2.7	2.3
CVS Asia	13.48	10:33:06	-12.41	94500	13.45	13.48	11000	12.38	18.34	15.39
Datio PLC	5.81	10:33:00	9.41	42500	5.8	5.81	1500	4.4	6.34	5.31
Dentems	4.87	10:33:05	0.2	12000	4.86	4.87	35500	3.91	5.67	4.86
ELE Manufacturing	7.19	10:33:01	-5.51	13500	7.19	7.2	51000	6.25	9.09	7.61

If you attempt to run a script that calls a push url - without first performing an asynchronous scan - the replay will halt while waiting for the response to the highlighted request. After two minutes, VuGen will display an error similar to the following, in the Replay log:

```
Action.c(140): Error -27782: Timeout (120 seconds) exceeded while
waiting to receive data for URL "http://push.example.com" [MsgId:
MERR-27782]
```

The error indicates that the response never finished.

Regenerating the script with Async Scan enabled will create a script similar to the following:

```
/* Added by Async CodeGen.
ID=Push_0
ScanType = Recording

The following urls are considered part of this conversation:
    http://push.example.com/STREAMING_IN_PROGRESS?LS_session=S343716d5eb050c64T2253451&LS_phase=4903&LS_domain=lights",

TODO - The following callbacks have been added to AsyncCallbacks.c.
Add your code to the callback implementations as necessary.
    Push_0_RequestCB
    Push_0_ResponseBodyBufferCB
    Push_0_ResponseCB
 */
    web_reg_async_attributes("ID=Push_0",
        "URL=http://push.example.com/STREAMING_IN_PROGRESS?LS_session={CorrelationParameter}&LS_phase=4903&LS_domain",
        "Pattern=Push",
        "RequestCB=Push_0_RequestCB",
        "ResponseBodyBufferCB=Push_0_ResponseBodyBufferCB",
        "ResponseCB=Push_0_ResponseCB",
        LAST);

    web_url("STREAMING_IN_PROGRESS",
        "URL=http://push.example.com/STREAMING_IN_PROGRESS?LS_session={CorrelationParameter}&LS_phase=4903&LS_domain=",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://www.app.example.com/GWT_StockListDemo_Basic/lightstreamer/lsengine.html",
        "Snapshot=t13.inf",
        "Mode=HTML",
        LAST);

    web_custom_request("control.js",
        "URL=http://push.example.com/lightstreamer/control.js", "",
        "Method=POST",
        "Resource=0",
        "RecContentType=text/plain",
        "Referer=http://push.example.com/ajax_frame.html?phase=594&domain=example.com&",
        "Snapshot=t14.inf",
        "Mode=HTML",
        "Body=LS_session={CorrelationParameter}&LS_table=1&LS_win_phase=19&LS_req_phase=311&LS_op=add&LS_mode=MERGE&LS_id=item1",
        LAST);
/* Added by Async CodeGen.
ID = Push_0
 */
    web_stop_async("ID=Push_0",
        LAST);


    return 0;
}
```

Notice that a **web_reg_async_attributes** function has been added before the **web_url** function that starts the *push* conversation, and that a **web_stop_async** function has been added after the last action step in the script. The script will now run successfully. The *push* conversation will remain active – running in parallel with the other script functions – until the **web_stop_async** function, or until the end of the script is reached.

Note that during the Async scan, VuGen did not remove (comment-out) any of the generated code in the Vuser script.

Asynchronous Example - Long-Poll

The following example describes a Vuser script that emulates an application that implements a *long-poll* asynchronous conversation. The application is a demo of a “chat” page. A browser shows the chat page, and sends a request that remains open until a new message is sent to the chat by another user. After such a message is sent:

- The response is finished.
- The new message is shown in the browser.
- The browser sends another request in order to listen for the next message sent to the chat.

Note: You can modify VuGen's asynchronous request thresholds to assist VuGen in finding long-poll type conversations. For details, see "Using Asynchronous Request Thresholds" on page 184.

The following is the Vuser script that VuGen generated after recording the application - before an asynchronous scan was performed. The script contains a series of **web_url** functions with similar URLs. Since new requests are sent as soon as the previous response is finished, no **lr_think_time** functions are added between the **web_url** functions. This help to indicate that this is a *long-poll* conversation and not a *poll* conversation.

When the Vuser script runs, requests to the chat application should be sent repeatedly every time a response from the chat application is finished. In addition, requests should be sent in parallel (simultaneously) with other actions performed in the script.

After VuGen performs an asynchronous scan on the script, the modified script looks as follows:

```
/* Added by Async CodeGen.
ID=LongPoll_0
ScanType = Recording

The following urls are considered part of this conversation:
    http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
    http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
    http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
    http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%

TODO - The following callbacks have been added to AsyncCallbacks.c.
Add your code to the callback implementations as necessary.
    LongPoll_0_RequestCB
    LongPoll_0_ResponseCB
- */
    web_reg_async_attributes("ID=LongPoll_0",
        "URL=http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
        "Pattern=LongPoll",
        "RequestCB=LongPoll_0_RequestCB",
        "ResponseCB=LongPoll_0_ResponseCB",
        LAST);

    web_url("request.ashx_3",
        "URL=http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t4.inf",
        "Mode=HTML",
        LAST);

/* Removed by Async CodeGen.
ID = LongPoll_0
- */
    /*  http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
 web_url("request.ashx_4",
        "URL=http://example.com/request.ashx?key=111111-1111-1111-1111-111111token=123488858&message=%5B%
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t5.inf",
        "Mode=HTML",
        LAST);
-   */

/* Removed by Async CodeGen.
```

Notice that a **web_reg_async_attributes** function has been added before the first **web_url** function that calls the chat application.

Except for the first call the chat application, all other **web_url** functions that call similar URLs have been commented out.

A close look at the Snapshot pane for the remaining **web_url** function shows that the snapshots for the removed steps now have origin = Polling. Notice that the response times vary greatly as the responses arrive only when another user has sent a chat message. This helps to indicate that this is a long-poll conversation, and not a poll conversation.

Preparing Scripts for Load Testing

General Vuser Functions

The general Vuser functions are also called LR functions because each LR function has an **lr** prefix. The LR functions can be used in any type of Vuser script. The LR functions enable you to:

- Get run-time information about a Vuser, its Vuser Group, and its host.

- Add transactions and synchronization points to a Vuser script. For example, the **lr_start_transaction** (**lr.start_transaction** in Java) function marks the beginning of a transaction, and **the lr_end_transaction** (**lr.end_transaction** in Java) function marks the end of a transaction. See "Preparing Scripts for Load Testing" on previous page for more information.
- Send messages to the output, indicating an error or a warning.

For details see the *Function Reference* (**Help > Function Reference**).

Protocol-Specific Vuser Functions

In addition to the general Vuser functions, VuGen also generates and inserts protocol-specific functions into the Vuser script while you record.

The protocol-specific functions are particular to the type of Vuser that you are recording. For example, VuGen inserts LRD functions into a database script, LRT functions into a Tuxedo script, and LRS functions into a Windows Sockets script.

By default, VuGen's automatic script generator creates Vuser scripts in C for most protocols, and in Java for Java type protocols. You can instruct VuGen to generate code in Visual Basic or Javascript. For more information, see "General > Script Node" on page 292.

All standard conventions apply to the scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other programming languages.

The following segment from a Web Vuser script shows several functions that VuGen recorded and generated in a script:

```
#include "as_web.h"
Action1()
{
    web_add_cookie("nav=140; DOMAIN=dogbert");
    web_url("dogbert",
        "URL=http://dogbert/",
        "RecContentType=text/html",
        LAST);
    web_image("Library",
        "Alt=Library",
        LAST);
    web_link("1 Book Search:",
        "Text=1 Book Search:",
        LAST);
    lr_start_transaction("Purchase_Order");
...
```

For more information about using C functions in your Vuser scripts, see the Function Reference (Help > Function Reference). For more information about modifying a Java script, see "Java Protocol - Manually Programming Scripts" on page 544.

Note: The C Interpreter used for running Vuser scripts written in C, only supports the ANSI C language. It does not support the Microsoft extensions to ANSI C.

Password Encoding

You can encode passwords in order to use the resulting strings as arguments in your script or parameter values. For example, your Web site may include a form in which the user must supply a password. You may want to test how your site responds to different passwords, but you also want to protect the integrity of the passwords. The **Password Encoder** enables you to encode your passwords and place secure values into the table.

To encode a password, select **Start > All Programs > HP Software > HP LoadRunner > Tools > Password Encoder**.

For task details, see "How to Encode a Password" on next page.

For user interface details, see "Password Encoder Dialog Box" on page 211.

Encrypting Text

You can encrypt text within your script to protect your passwords and other confidential text strings. You can perform encryption both automatically, from the user interface, and manually, through programming. You can restore the string at any time, to determine its original value. When you encrypt a string, it appears in the script as a coded string. Note that VuGen uses 32-bit encryption.

In order for the script to use the encrypted string, it must be decrypted with **lr_decrypt**.

```
lr_start_transaction(lr_decrypt("3c29f4486a595750"));
```

For task details, see "How to Encrypt/Decrypt Text" on next page.

Transaction Overview

You define *transactions* to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified Vuser requests. These requests can be simple tasks such as waiting for a response for a single query, or complex tasks, such as submitting several queries and generating a report.

To measure a transaction, you insert Vuser functions to mark the beginning and the end of a task. Within a script, you can mark an unlimited number of transactions, each transaction with a different name.

For LoadRunner, the Controller measures the time that it takes to perform each transaction. After the test run, you analyze the server's performance per transaction using the Analysis' graphs and reports.

Before creating a script, you should determine which business processes you want to measure. You then mark each business process or sub-process as a transaction.

Transaction names cannot contain a "_" or "@" symbol. This will cause errors to occur when attempting to open the Analysis Cross Results graphs.

You can create transactions either during or after recording. For task details, see "How to Insert Transactions" on page 204.

Rendezvous Points

When performing load testing, you need to emulate heavy user load on your system. To accomplish this, you synchronize Vusers to perform a task at exactly the same moment. You configure multiple Vusers to act simultaneously by creating a rendezvous point. When a Vuser arrives at the rendezvous point, it waits until all Vusers participating in the rendezvous arrive. When the designated number of Vusers arrive, the Vusers are released.

You designate the meeting place by inserting a rendezvous point into your Vuser script. When a Vuser executes a script and encounters the rendezvous point, script execution is paused and the Vuser waits for permission from the Controller to continue. After the Vuser is released from the rendezvous, it performs the next task in the script.

For task details, see "How to Prepare a Script for Load Testing" on page 205.

Note: Rendezvous points are only effective in Action sections—not init or end.

How to Encrypt/Decrypt Text

This task describes how to encrypt and decrypt strings in your code. For background information, see "Encrypting Text" on previous page.

Encrypt a string

1. Select the text you want to encrypt.
2. Select **Encrypt string (*string*)** from the right-click menu.

Restore an encrypted string

1. Select the string you want to restore.
2. Select **Restoreencrypted string (*string*)** from the right-click menu.

   For more information on the **lr_decrypt** function, see the Function Reference (**Help > Function Reference**).

How to Encode a Password

This task describes how to encode a password. You can encode passwords in order to use the resulting strings as arguments in your script or parameter values. For example, your Web site may include a form in which the user must supply a password. You may want to test how your site responds to different passwords, but you also want to protect the integrity of the passwords.

Encode a password

1. From the Windows menu, select **Start > All Programs > HP Software > HP LoadRunner > Tools > Password Encoder**. The Password Encoder dialog box opens.
2. Enter the password in the **Password** box.
3. Click **Generate**. The Password Encoder encrypts the password and displays it in the **Encoded String** box.

4. Use the **Copy** button to copy and paste the encoded value into the Data Table.

How to Create a Controller Scenario from VuGen

Note: The following section only applies to LoadRunner. For information on integrating scripts into Business Process profiles, see the HP Business Service Management documentation.

Normally, you create a scenario from the LoadRunner Controller. You can also create a basic scenario from VuGen using the current script.

To create this type of scenario, select **Tools** > **Create Controller Scenario** and complete the dialog box. For user interface details, see "Create Controller Scenario Dialog Box" on page 210.

For more information, see the HP Controller User Guide.

How to Insert Transactions

Notes:

- You can create *nested* transactions—transactions within transactions. If you nest transactions, close the inner transactions before closing the outer ones—otherwise the transactions won't be analyzed properly. However, transactions must be contained within a single **action** section.
- Transaction names must be unique and begin with a letter or number and may contain letters or numbers. Do not use the following characters: . , : # / \ " <.

The following steps describe different methods to insert transactions. For background information, see "Transaction Overview" on page 202.

Insert Transactions in the Editor

- To mark the start of a transaction after recording, select **Design > Insert in Script > Start Transaction** and enter a transaction name.
- To mark the end of a transaction, select **Design > Insert in Script > End Transaction**. VuGen inserts an **lr_end_transaction** statement into the Vuser script.

Insert Transactions During Recording

- To mark the start of a transaction, click the **Start Transaction** button  on the Recording toolbar while recording a script and enter a transaction name. When you click **OK**, VuGen inserts an **lr_start_transaction** statement into the Vuser script.
- To mark the end of a transaction, click the **End Transaction** button  on the Recording toolbar and select the transaction to close. When you click **OK**, VuGen inserts an **lr_end_transaction** statement into the Vuser script.

How to Display Transactions

The following steps describe how to display different types of transactions when viewing them in the task pane. For background information, see "Transaction Overview" on page 202.

Display Hidden Transactions

To display the hidden transactions—the non-primary and client side transactions—click the button adjacent to **Show hidden transactions** at the bottom of the transaction list. VuGen lists the hidden transactions in gray. To hide them, click the button again.

Display Transactions With Errors

Transactions with errors are those that do not measure any server steps, or those with illegal names. To show the transactions with errors, click the **Show transactions with errors** button. VuGen lists the transactions with errors in red. To hide them, click the button again.

Display Transactions for Non-primary Steps

To show the transactions for non-primary steps, you need to display all of the thumbnails. Select **View > Show All Thumbnails**.

How to Prepare a Script for Load Testing

This task describes the additional things you can do to your script after you have debugged it to prepare it for load testing. All of the items in this task are optional.

Search Steps

You can use the **Step Navigator** pane to search for specific steps in your script based on a variety of criteria. This can be helpful if you have a script with a lot of steps.

Enter a search string and select the part of the steps you want to search. The steps pane dynamically filters the steps in the display for those that match your search criteria. For user interface details, see " Step Navigator Pane" on page 93.

Insert Steps

You can insert a variety of steps into your script such as think time steps, debug messages, and output messages. For task details, see "How to Insert Steps into a Script" on page 207.

Edit Steps

You can modify individual step properties by selecting **Show Arguments** from the following ways :

- Right-clicking the step in the script editor
- Right-clicking the icon within the gutter of the script editor
- Select the step in the Steps Navigator pane

Insert Transactions

You can insert transactions into your script by using the **Design > Insert in Script > Start Transaction** and **Design > Insert in Script > End Transaction** menu items. For task details, see "How to Insert Transactions" on previous page.

Insert Rendezvous Points

You can synchronize Vusers to perform a task at exactly the same moment by creating a rendezvous point. When a Vuser arrives at the rendezvous point, it waits until all Vusers participating in the rendezvous arrive. When the designated number of Vusers arrive, the Vusers are released.

You can insert rendezvous points in one of the following ways:

- To insert a rendezvous point while recording, click the **Rendezvous** button  on the Recording toolbar and enter a name in the dialog box (not case sensitive).
- To insert a rendezvous point after recording, select **Design > Insert in Script > Rendezvous** and enter a name for the rendezvous point (not case sensitive).

When a rendezvous point is inserted, VuGen inserts a **lr_rendezvous** function into the Vuser script. For example, the following function defines a rendezvous point named rendezvous1:

```
lr_rendezvous("rendezvous1");
```

For concept details, see "Rendezvous Points" on page 203.

Insert Comments

VuGen allows you to insert comments between Vuser activities. You can insert a comment to describe an activity or to provide information about a specific operation. For example, if you are recording database actions, you could insert a comment to mark the first query, such as "This is the first query."

You can insert a comment in one of the following ways:

- To insert a comment while recording, click the **Insert Comment** button  on the Recording toolbar and enter the desired comment in the Insert Comment dialog box.
- To insert a comment after recording, select **Design > Insert in Script > Comment** and enter the comment.

The following script segment shows how a comment appears in a Vuser script:

```
/* <comments> */
```

Insert VuGen Functions

You can insert VuGen functions at this point. For a list of some useful functions see "Useful VuGen Functions" on page 208.

Insert Log Messages

You can use VuGen to generate and insert **lr_log_message** functions into a Vuser script. For example, if you are recording database actions, you could insert a message to indicate the first query, "This is the first query."

To insert a log message, select **Design > Insert in Script > Log Message** and enter the message.

Add Files to the Script

You can add files to your script folder to make them available when running the script. If the files are text-based, you will be able to view and edit them in VuGen's editor.

To add a file to the script, select **File** > **Add Files to Script**.

Synchronize the Script (RTE Vusers only)

You can add synchronization functions to synchronize the execution of the Vuser script with the output from your application. Synchronization applies to RTE Vuser scripts only.

The following is a list of the available synchronization functions:

Function	Description
TE_wait_cursor	Waits for the cursor to appear at a specified location in the terminal window.
TE_wait_silent	Waits for the client application to be silent for a specified number of seconds.
TE_wait_sync	Waits for the system to return from X-SYSTEM or Input Inhibited mode.
TE_wait_text	Waits for a string to appear in a designated location.
TE_wait_sync_ transaction	Records the time that the system remained in the most recent X-SYSTEM mode.

For details about synchronization in RTE Vuser scripts, see "RTE Synchronization Overview" on page 622.

Configure the Test Results Window Options

To assist with debugging a Vuser script, you can view a report that summarizes the results of your script run. generates the report during the Vuser script execution and you view the report when script execution is complete.

By default, VuGen generates test results and automatically opens them at the end of a run.

To prevent VuGen from generating the results, choose**Tools** > **Options** > **Script** , select the **Replay** tab and clear the **Generate report during script execution option**.

To indicate whether or not to open the results after running the script, choose **Tools** > **Options** > **Script** , select the **Replay** tab, and select a view in the **After Replay** section.

Replay and Debug your Script

For more information, see "Replaying and Debugging Vuser Scripts" on page 121.

How to Insert Steps into a Script

The following steps describe how to add different types of steps into a Vuser script.

Insert Think Time Steps

The time that a user waits between performing successive actions is known as the *think time*. Vusers use the **lr_think_time** function to emulate user think time. When your record a Vuser script, VuGen records the actual think times and inserts appropriate **lr_think_time** statements into the Vuser script. You can edit the recorded **lr_think_time** statements, and manually add more **lr_**

think_time statements to a Vuser script.

To add a think time step, select **Design > Insert in Script > New Step** >**Think Time** and specify the desired think time in seconds.

Insert Debug Messages

You can add a debug or error message using VuGen's user interface. For debug messages you can indicate the level of the text message—the message is only issued when your specified level matches the message class. You set the message class using **lr_set_debug_message**.

To insert a debug message, select **Design> Insert in Script > New Step** > **Debug Message** and complete the dialog box. For user interface details, see Debug Message Dialog Box.

Insert Error and Output Messages

For protocols that support the **Step Navigator**, such as Web, Winsock, and Oracle NCA, you can add an error or output message using the user interface. A common usage of this function is to insert a conditional statement, and issue a message if the error condition is detected.

To insert an error or output message, select **Design > Insert in Script > New Step** > **Error Message** or **Output Message,** and enter the message. An **lr_error_message** or **lr_output_ message** function is inserted at the current point in the script.

Useful VuGen Functions

This section contains useful VuGen functions that you may want to add to your script while debugging or preparing your script for load testing.

Obtaining Vuser Information

You can add the following functions to your Vuser scripts to retrieve Vuser information:

Function	Description
lr_get_attrib_ string	Returns a command line parameter string.
lr_get_host_ name	Returns the name of the machine running the Vuser script.
lr_get_master_ host_name	Returns the name of the machine running the Controller. Not applicable when working with the HP Business Service Management.
lr_whoami	Returns the name of a Vuser executing the script. Not applicable when working with the HP Business Service Management.

In the following example, the **lr_get_host_name** function retrieves the name of the computer on which the Vuser is running.

```
my_host = lr_get_host_name( );
```

For more information about the above functions, see the Function Reference (**Help > Function Reference**).

Sending Messages to Output

Using the Message type functions in your Vuser script, you can send customized error and notification messages to the output and log files, and to the Test Report summary. For example, you could insert a message that displays the current state of the client application. The LoadRunner Controller displays these messages in the Output window. You can also save these messages to a file.

When working with HP Business Service Management, you can use Message type functions to send error and notification messages to the Web site or Business Process Monitor log files. For example, you could insert a message that displays the current state of the Web-based application.

Note: Do not send messages from within a transaction as this may lengthen the transaction execution time and skew the transaction results.

You can use the following message functions in your Vuser scripts:

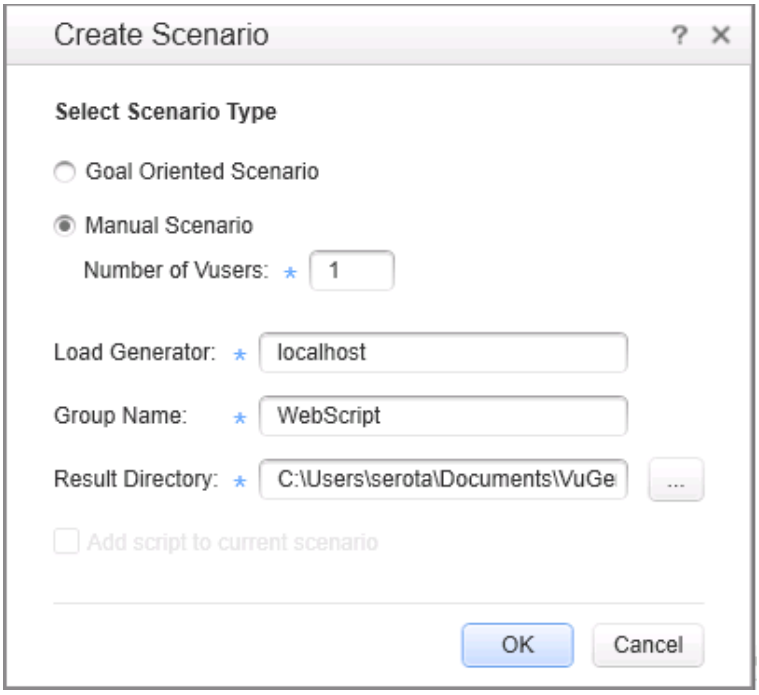
Function	Description
lr_debug_message	Sends a debug message to the Output window or the Business Process Monitor log file.
lr_error_message	Sends an error message to the Output window, Test Results report, or the Business Process Monitor log files.
lr_get_debug_message	Retrieves the current message class.
lr_log_message	Sends an output message directly to the log file, *output.txt*, located in the Vuser script folder. This function is useful in preventing output messages from interfering with TCP/IP traffic.
lr_output_message	Sends a message to the Output window, Test Results report, or the Business Process Monitor log files.
lr_set_debug_message	Sets a message class for output messages.
lr_vuser_status_message	Sends a message to the Vuser status area in the Controller. Not applicable when working with the HP Business Service Management.
lr_message	Sends a message to the Vuser log and Output window or the Business Process Monitor log files.

The behavior of the **lr_message**, **lr_output_message**, and **lr_log_message** functions are not affected by the script's debugging level in the Log run-time settings—they will always send messages.

Using the **lr_output_message**, and **lr_error_message** functions, you can also send meaningful messages to the Test Results summary report. For information, see "Viewing Test Results" on page 214

Create Controller Scenario Dialog Box

This dialog box enables you to create a basic Controller scenario from within VuGen.



To access	**VuGen > Tools > Create Controller Scenario**
Relevant tasks	"How to Create a Controller Scenario from VuGen" on page 204

User interface elements are described below:

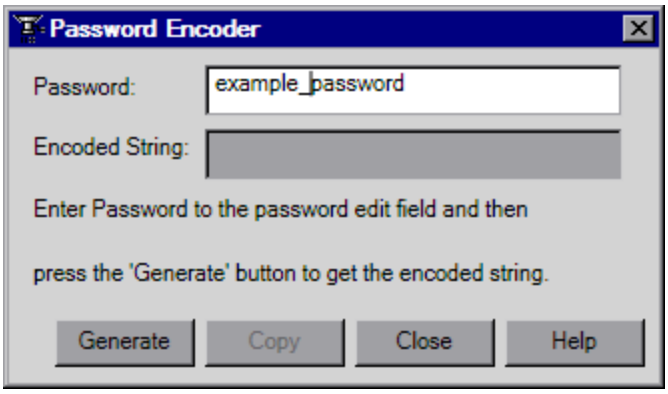
UI Element	Description
Add script to current scenario	If a scenario is currently open in the Controller and you want to add the script to this scenario, select this check box. If you clear the check box, LoadRunner opens a new scenario with the specified number of Vusers.
Group Name	For a manual scenario, users with common traits are organized into groups. Specify a new group name for the Vusers.
Load Generator	The name of the machine that will run the scenario.
Results Directory	Enter the desired location for the results.
Script Name	For a goal-oriented scenario, specify a script name.

, continued

Select Scenario Type	• **Goal Oriented Scenario.** LoadRunner automatically builds a scenario based on the goals you specify. • **Manual Scenario.** The scenario is created manually by specifying the number of Vusers to run.

Password Encoder Dialog Box

This dialog box enables you to generate encoded passwords.



To access	**Start > All Programs > HP Software > HP LoadRunner > Tools > Password Encoder**
Relevant tasks	"How to Encode a Password" on page 203
See also	"Password Encoding" on page 202

User interface elements are described below:

UI Element	Description
Copy	Copy the results from the encoded string field to paste them to the Data table containing your list of parameters.
Encoded String	The encoded results are displayed here.
Generate	Click this to generate the encoded password.
Password	Enter the password you want to encode here.

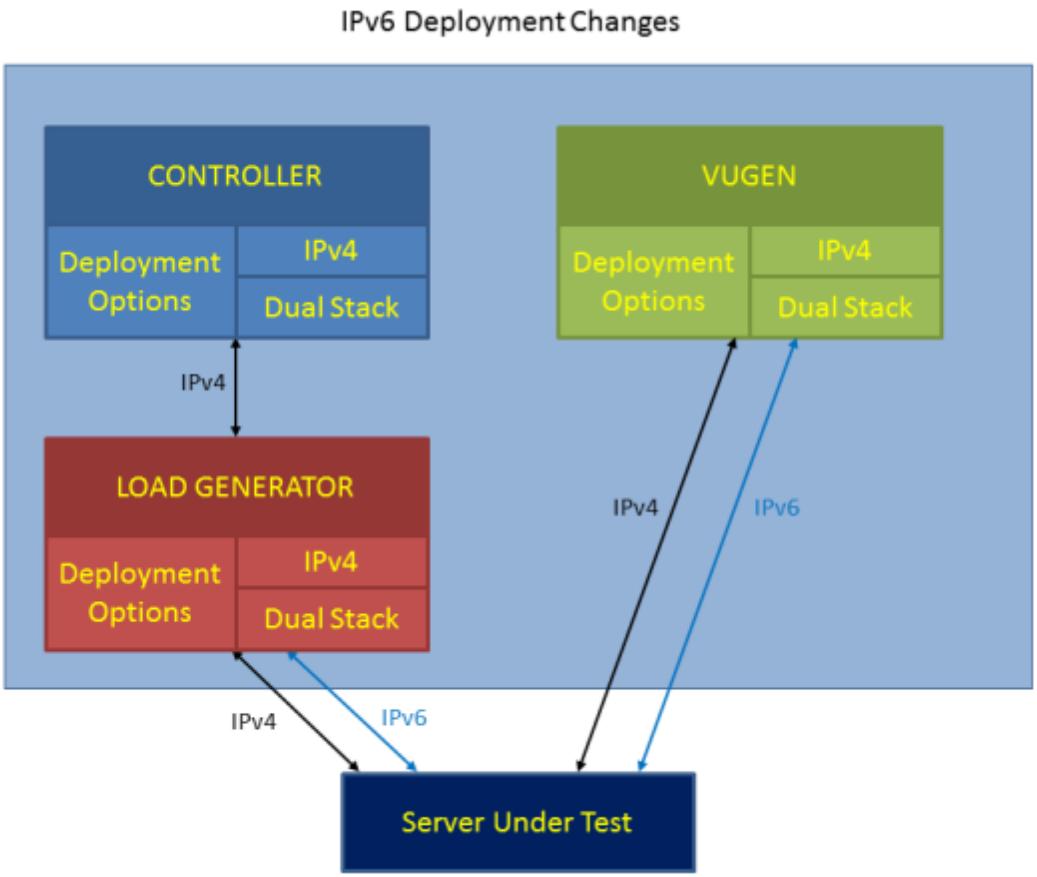
IPv6 Support

As IPv6 implementation becomes more widespread, Virtual User Generator now enables you to test IPv6 based applications in addition to IPv4 based ones. Script recording supports recording for both IPv4 and IPv6 simultaneously. The code that is generated is non-IP specific. Except for Web

HTTP based protocols, users will be unaware of which IP version is being used when replaying the script in a load test. Web HTTP protocols have a Runtime setting that allows you to choose between IPv4 and IPv6 for the replay.

IPv6 Deployment

The internal Virtual User Generator communication between the Controller and Load Generators still uses IPv4 communication. To record and replay in both IPv4 and IPv6, install both VuGen and Load Generator on IPv6-enabled computers, as shown in the diagram below.



For more details about IPv6 related changes, see Advanced Options Dialog Box and the IP Wizard section in the *Controller User Guide*.

Protocols Supported

The following protocols support IPv6 testing:

- AMF
- Citrix Nfuse
- DNS
- Flex
- FTP

- IMAP
- Java Over HTTP
- Mobile HTTP
- Mobile Ajax TruClient
- Multi Winsock
- NDM Webtrace
- Oracle NCA
- POP3
- RDP
- Silverlight
- SMTP
- Ajax TruClient
- Click and Script family
- Web (HTTP/HTML)
- Web Services

Protocol Support Limitations

Support for IPv6 is available for the protocols shown in the list above with the following limitations:

- **Web HTTP protocol**
  - FTP from Web is not supported
  - Web Breakdown is not supported
  - Kerberos is not supported
  - Spoofing from Web is not supported
  - PAC file is not supported
  - An explicit IP (in IPv6 format) in a URL argument cannot be used. For example, the following step will fail in replay:

    ```
    web_url("IPv6",
        "URL=http://[2001:0db8:85a3:0000:0000:8a2e:0370:7334]/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
    ```

- Webtrace
  - IPv6 Webtrace is not supported on 6to4 outgoing network interfaces.
  - IPv6 Webtrace is not supported on Windows XP and Windows 2003 operating systems. This is because those operating systems do not provide an API to access IPv6 routing & forwarding table..
  - The Raw Socket mode for IPv6 Webtrace is not supported; therefore you cannot trace routes to IPv6 targets by means of Raw Socket transport. This is because of Windows network protocol stack limitations for listening on SOCK_RAW type sockets of the AF_INET6 family,
- **General limitations**
  - Replay failures may occur because of a IPv4/IPv6 switch between recording and replaying.

Viewing Test Results

Test Results Overview

To assist with debugging a Vuser script, you can view a report that summarizes the results of your script run. VuGen generates the report during the Vuser script execution and you view the report when script execution is complete.

The Test Results window displays all aspects of the test run and can include:

- a high-level results overview report (pass/fail status)
- the data used in all runs
- an expandable tree of the steps, specifying exactly where application failures occurred
- the exact locations in the script where failures occurred
- a still image of the state of your application at a particular step
- a movie clip of the state of your application at a particular step or of the entire test
- detailed explanations of each step and checkpoint pass or failure, at each stage of the test

Customizing the Test Results Display

Each result set is saved in a single **.xml** file (called **results.xml**). This **.xml** file stores information on each of the test result nodes in the display. The information in these nodes is used to dynamically create **.htm** files that are shown in the top-right pane of the Test Results window.

Each node in the run results tree is an element in the **results.xml** file. In addition, there are different elements that represent different types of information displayed in the test results. You can take test result information from the **.xml** file and use XSL to display the information you require in a customized format (either when printing from within the Test Results window, when displaying test results in your own customized results viewer, or when exporting the test results to an HTML file).

XSL provides you with the tools to describe exactly which test result information to display and exactly where and how to display, print or export it. Using a XSL editor, you can modify the **.css** and **.xsl** files in the results folder, to change the appearance of the report (fonts, colors, and so forth).

For example, in the **results.xml** file, one element tag contains the name of an action, and another element tag contains information on the time at which the run was performed. Using XSL, you could tell your customized editor that the action name should be displayed in a specific place on the page and in a bold green font, and that the time information should not be displayed at all.

Connecting to Application Lifecycle Management from the Test Results Window

To manually submit defects to Application Lifecycle Management from the Test Results window, you must be connected to Application Lifecycle Management.

The connection process has two stages.

- First, you connect to a local or remote Application Lifecycle Management server. This server handles the connections between the Test Results and the Application Lifecycle Management project.
- Next, you log in and choose the project you want to access. The project stores tests and run session information for the application you are testing.

Note: Application Lifecycle Management projects are password protected, so you must provide a user name and a password.

For more information on connecting to an ALM project, see "How to Work with Scripts in ALM Projects" on page 223.

How to Send Custom Information to the Report

In addition to the information sent automatically to the report, for Web Service Vusers, you can send information to the report using the message functions **lr_output_message** or **lr_error_message**.

For task details, see "How to Insert Steps into a Script" on page 207.

How to Configure the Appearance of the Test Results Window

By default, the Test Results window has the same look and feel as the window, using the Microsoft Office 2003 theme. You can change the look and feel of the Test Results window, as required.

To change these settings, select **View > Window Theme** and select the desired theme.

Note: You can apply the Microsoft Windows XP theme to the Tests Results window only if your computer is set to use a Windows XP theme.

How to Open the Test Results of a Specific Run

This task describes how to open the test results window for a specific run.

1. Select **File > Open** from within the Test Results window.
2. Select a script file to display the test results for that file and select the desired test result file. By default, result files for tests are stored in **<Script>\<ResultName>.xml**. If your script is stored in Application Lifecycle Management, see below.
3. Select a results set and click **Open**.

Note: Results files for earlier versions were saved with a **.qtp** file extension. In the Select Results File dialog box, only results files with an **.xml** extension are shown by default. To view results files with a **.qtp** extension in the Select Results File dialog box, select **Test Results (*.qtp)** in the **Files of type** box.

Select a Script Stored in Application Lifecycle Management

1. Click the **Application Lifecycle Management Connection** button  and connect to your Application Lifecycle Management project.
2. In the Open Test Results dialog box, enter the path of the folder that contains the results file for your test, or click the browse button to open the Open Test from ALM Project dialog box.
3. Select **DB Vuser** in the **Test Type** list.
4. Select the script whose test results you want to view, and click **OK**.
5. In the Open Test Results dialog box, highlight the test result set you want to view, and click **Open**. The Test Results window displays the selected test results.

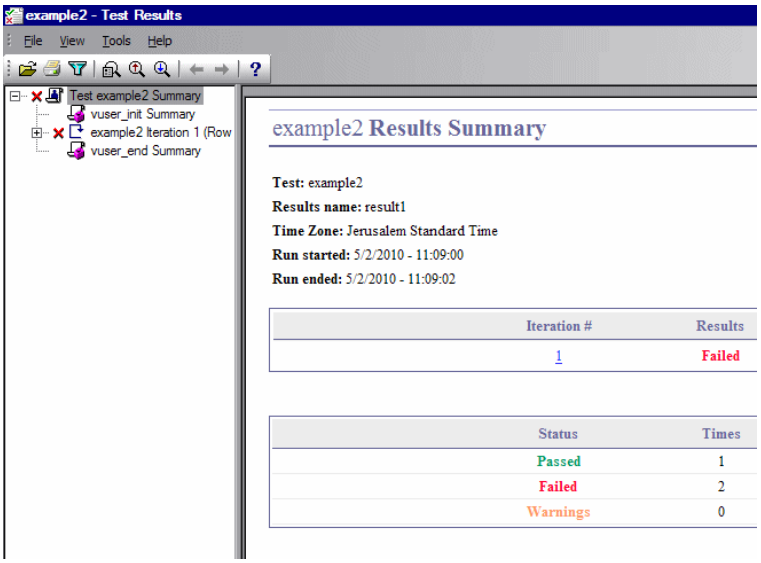
How to Find Steps in the Test Results

This task describes how to search the test results for steps of a particular type.

1. Select **Tools > Find** from within the Test Results window.
2. Select the type of step you wish to find. You can select multiple options.
3. Select **Up** or **Down** to indicate the direction of the search.
4. Select **Find Next** to find the next occurrence of the type of step you selected.

Test Results Window

This window displays a report that summarizes the results of your script run.



To access	• **Opens automatically after running a script.** • **VuGen > Replay > Test Results...**
Important information	• You can configure the result settings from **Tools > Options > Scripting > Replay** . • The Test Results window can show results with up to 300 levels in the tree hierarchy. If you have results with more than 300 nested levels, you can view the entire report by manually opening the **results.xml** file.

User interface elements are described below:

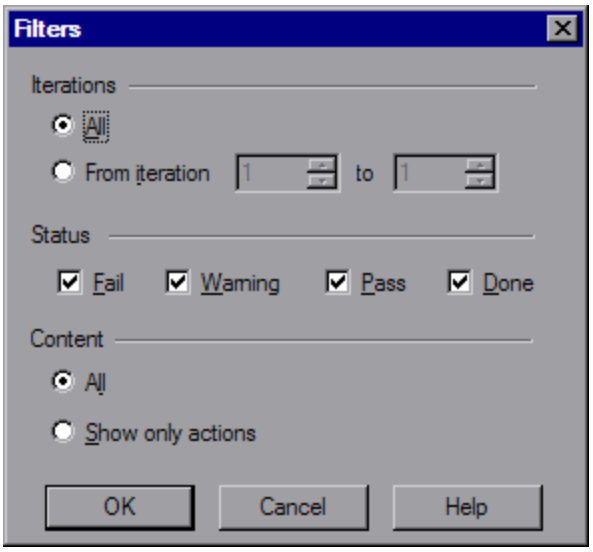
UI Element	Description
Report Tree	A graphical representation of the test results located in a pane on the left of the window. You can collapse or expand a branch in the run results tree to change the level of detail that the tree displays. The icons next to the steps indicate the following information: ✔ Indicates a step that succeeded. ✘ Indicates a step that failed. ! Indicates a warning, meaning that the step did not succeed, but it did not cause the test to fail. ! ⛔ Indicates a step that failed unexpectedly. You can expand and collapse all of the nodes from the **View** menu, or by clicking ***.**

, continued

Results Details Pane (Results Summary)	Contains details of the test run which change depending on which part of the report tree you select. When you select the top node of the tree, the Result Details tab shows a summary of the results for your test. When you select a branch or step in the tree, the Result Details tab contains the details for that step. The Result Details tab may also include a still image of your application for the highlighted step. • Iterations, actions, and steps that contain checkpoints are marked **Passed** or **Failed** in the right part of the Test Results window and are identified by icons in the tree pane. • Iterations, actions, and steps that ran successfully, but do not contain checkpoints, are marked **Done** in the right part of the Test Results window.
Screen Recorder Tab	Contains the movie associated with your test results. If there is no movie associated with your test results, the Screen Recorder tab contains the message: `No movie is associated with the results.`
	Opens the test results of a specific run. For more details, see "How to Open the Test Results of a Specific Run" on page 215.
	Opens the Print dialog box, enabling you to print the test results. For more information, see "Print Dialog Box" on next page.
	Opens the Filters dialog box, enabling you to filter the test results. For more information, see "Filters Dialog Box" below.
	Opens the Find dialog box, enabling you to search the test results for steps of a particular type. For more information, see "How to Find Steps in the Test Results" on page 216.
	Searches the test results for the previous step matching the criteria in the Find dialog box.
	Searches the test results for the next step matching the criteria specified in the Find dialog box.
	Selects the previous node.
	Selects the next node.
	Opens the product documentation.

Filters Dialog Box

This dialog box enables you to filter the test results in the test results window.



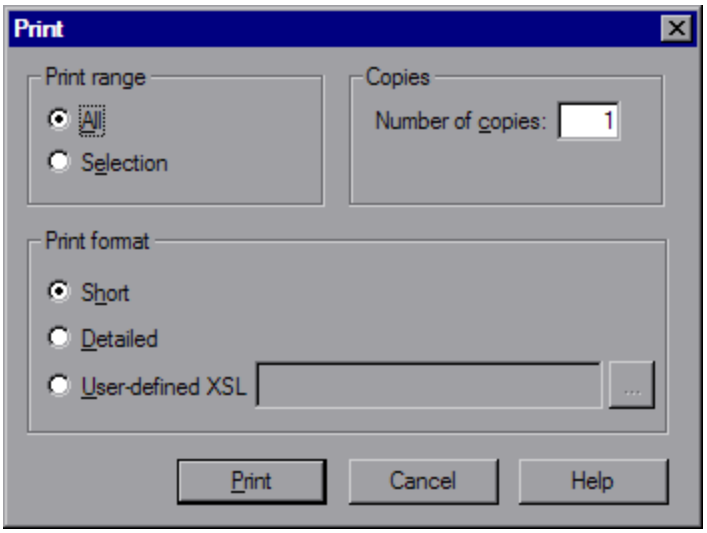
To access	"Test Results Window" on page 216 **> View > Filters**

User interface elements are described below:

UI Element	Description
Iterations	• **All.** Displays test results from all iterations. • **Fromiteration X to Y.** Displays the test results from a specified range of test iterations.
Status	• **Fail.** Displays the results for the steps that failed. • **Warning.** Displays the results for the steps with the status **Warning** (steps that did not pass, but did not cause the script to fail). • **Pass.** Displays the results for the steps that passed. • **Done.** Displays the results for the steps with the status **Done** (steps that were performed successfully but did not receive a pass, fail, or warning status).
Content	• **All.** Displays all steps from all nodes in the test. • **Show only actions.** Displays the action nodes in the test (not the specific steps in the action nodes).

Print Dialog Box

This dialog box enables you to print the test results.



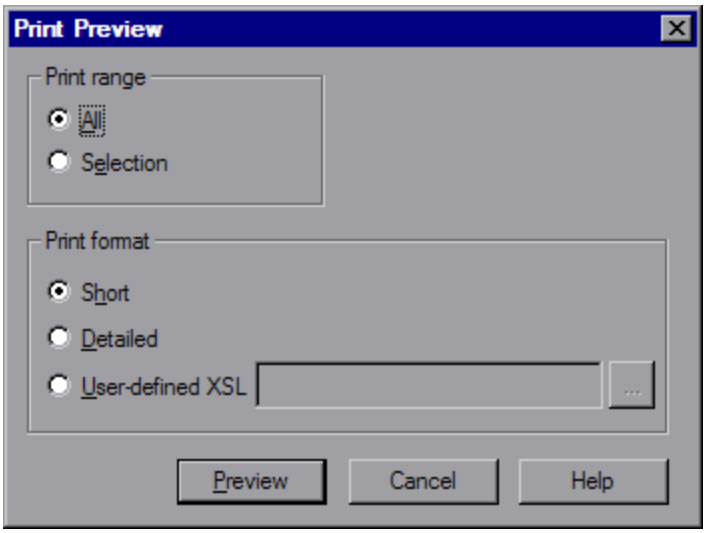
To access	"Test Results Window" on page 216 **> File > Print**

User interface elements are described below:

UI Element	Description
Print range	• **All.** Prints the results for the entire script. • **Selection.** Prints the results for the selected branch in the run results tree.
Copies	The number of copies to print.
Print format	• **Short.** Prints a summary line (when available) for each item in the run results tree. This option is available only if you the print range is set to **All**. • **Detailed.** Prints all available information for each item in the run results tree, or for the selected branch. • **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 214.

Print Preview Dialog Box

This dialog box enables you to view a print preview of the test results.



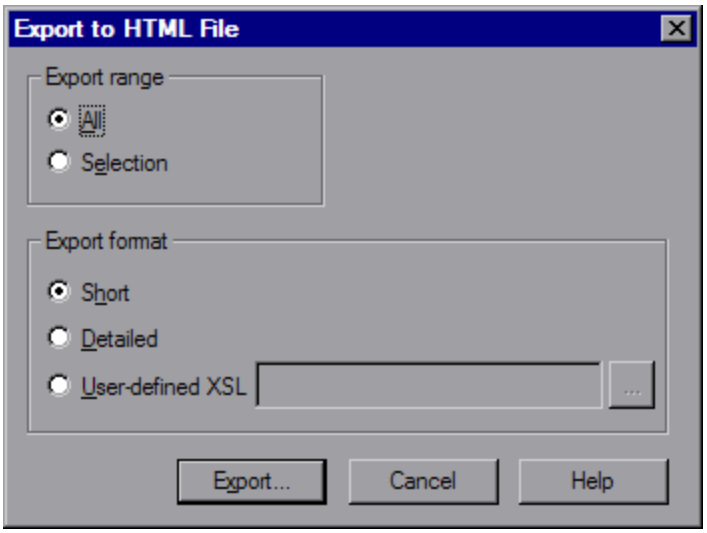
To access	"Test Results Window" on page 216 **> File > Print Preview**
Important information	If some of the information is cut off in the preview, for example, if checkpoint names are too long to fit in the display, click the **Page Setup** button in the Print Preview window and change the page orientation from **Portrait** to **Landscape**.

User interface elements are described below:

UI Element	Description
Print range	• **All.** Prints the results for the entire script. • **Selection.** Prints the results for the selected branch in the run results tree.
Print format	• **Short.** Prints a summary line (when available) for each item in the run results tree. This option is available only if you the print range is set to **All**. • **Detailed.** Prints all available information for each item in the run results tree, or for the selected branch. • **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 214.

Export to HTML File Dialog Box

This dialog box enables you to export the test results to an HTML file. This enables you to view the results even if the Test Results Viewer environment is unavailable. For example, you can send the file containing the results in an e-mail to a third-party. You can select the type of report you want to export, and you can also create and export a customized report.



To access	"Test Results Window" on page 216 > File > Export to HTML File

User interface elements are described below:

UI Element	Description
Export Range	• **All.** Exports the results for the entire script. • **Selection.** Exports result information for the selected branch in the results tree.
Export Format	• **Short.** Prints a summary line (when available) for each item in the run results tree. This option is available only if you the print range is set to **All**. • **Detailed.** Prints all available information for each item in the run results tree, or for the selected branch. • **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 214.

Working with Application Lifecycle Management

Managing Scripts Using ALM - Overview

VuGen works together with *HP Application Lifecycle Management* (ALM). ALM provides efficient functionality for storing and retrieving Vuser scripts, scenarios, and analysis results. You can store scripts in an ALM project and organize the scripts into unique groups.

In order for VuGen to access an ALM project, you must connect VuGen to the Web server on which the ALM project is located. You can connect to either a local or remote Web server.

For more information on working with ALM, see the *HP Application Lifecycle Management User Guide*.

ALM Version Control - Overview

VuGen supports version control features in Vuser scripts saved in ALM projects that use version control.

The version control features change the process of opening and saving a script. Scripts with version control are either in a state of checked-in or checked-out. When you are working with a script in a checked-out state, any changes you make will not be saved on the ALM server until you check in the script. If you save the script from within VuGen, a temporary file is saved on your machine that protects your changes in case your computer crashes.

If you are working with a script in a checked-in state, the script is read-only and you cannot make any changes until you check out the script.

If a particular script is being saved to ALM for the first time, and the project uses version control, the script automatically starts in a checked-out state.

How to Work with Scripts in ALM Projects

The following steps describe the workflow of how to work with Vuser scripts that are saved in an ALM project.

Note: To work with scripts in ALM projects with version control, see "How to Work with Version Controlled Scripts in ALM Projects" on next page.

1. **Connect to ALM**

   Open a connection to the ALM server and project that contains the script. For task details, see "How to Connect to ALM" below.

2. **Open the script**

   Select **File > Open > Script/Solution**. In the Open VuGen Script or Solution dialog box, select the script to open and then click **Open**.

3. **Save the script**

   Select **File > Save Script**. If the script is in a project that uses version control and is not checked out, the script is saved as a temporary file on your local machine.

How to Connect to ALM

To store and retrieve scripts from ALM, you need to connect to an ALM project. You can connect or disconnect from an ALM project at any time during the testing process.

You can connect to one version of HP ALM from VuGen and a different version from your browser. For more information, see the **Important Information** section in "HP ALM Connection Dialog Box [VuGen]" on page 226.

Connect to a project in ALM

1. Select **ALM** > **ALM Connection**. The HP ALM Connection dialog box opens.

2. Step 1: Enter the server URL, user name, and password and then click **Connect**. VuGen connects to the ALM server.

   To disconnect from ALM, click **Disconnect**.

3. Step 2: Enter the domain and project details, and then click **Login**. VuGen logs in to the specified project.

   To log out of the project, click **Logout**.

4. Click **Close** to close the HP ALM Connection dialog box.

How to Work with Version Controlled Scripts in ALM Projects

The following steps describe the workflow of how to work with scripts saved in ALM projects that use version control.

Note: This procedure is relevant only for scripts in ALM projects that support version control and have the Performance Center addition installed. If these two conditions are not met, see "How to Work with Scripts in ALM Projects" on previous page.

1. ### Connect to ALM

   Open a connection to the ALM server and project that contains the script. For task details, see "How to Connect to ALM" on previous page.

2. ### Open the script

   Select **File > Open > Script/Solution**. In the Open VuGen Script or Solution dialog box, select the script to open and then click **Open**.

3. ### Check in/out the script

   If the ALM project has version control, each script is always defined as being either checked-in or checked-out. For more details, see "ALM Version Control - Overview" on previous page. To check in and check out scripts, select **ALM > Check In/Check Out**.

   Note: If the ALM project has version control, the file is locked when it is checked out.

   If the ALM project is not version controlled, the file is not locked when checked out of the project.

4. ### Cancel a check out (optional)

   If you checked out a script and do not want to save the changes, you can return the status of the script to checked-in without saving by selecting **ALM > Undo Check Out**.

5. ### Save the script

   Select **File > Save Script**. If the script is in a project that uses version control and is not checked out, the script is saved as a temporary file on your local machine.

How to Save VuGen Vuser Scripts to ALM Projects

The following steps describe how to save a Vuser script to an ALM project.

1. **Open or create the Vuser script**

   Create or open the desired script in VuGen.

2. **Connect to ALM**

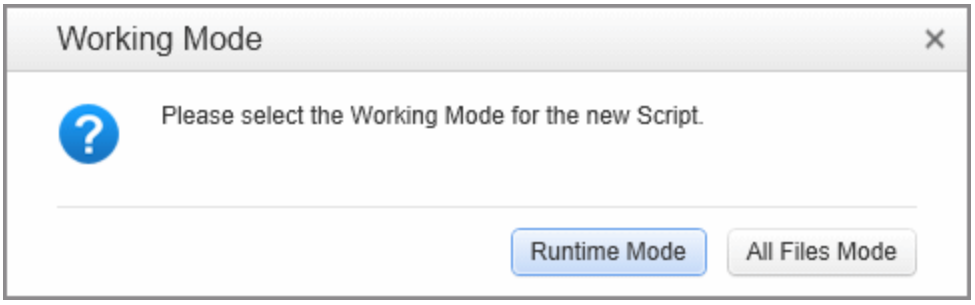
   Open a connection to the ALM server and project in which you want to store the script. For task details, see "How to Connect to ALM" on page 223.

3. **Save the script to ALM**

   a. Select **File > Save Script as**. The Save Script As dialog box opens.

   b. Click **ALM Test Plan**, and then specify the name and location for the script.

   c. Click **Save**. After a short time, the Working Mode dialog box opens.



   d. Select one of the following options:

   Runtime Mode. Copies only the files needed to replay the script. This option does not copy recording snapshot files and other unnecessary files. This results in a shorter transfer time.

   All Files Mode. Copies all of the files associated with this script. This results in a longer transfer time.

How to Compare Previous Versions of a Script

If your Vuser script is saved in an ALM project that uses version control, you can compare previous versions of the script. The following steps describe how to do this.

1. **Connect to ALM**

   Open a connection to the ALM server and project that contains the script that you want to view or modify. For task details, see "How to Connect to ALM" on page 223.

2. **Open the script**

   Select **File > Open > Script/Solution**. In the Open VuGen Script or Solution dialog box, select the script to open and then click **Open**.

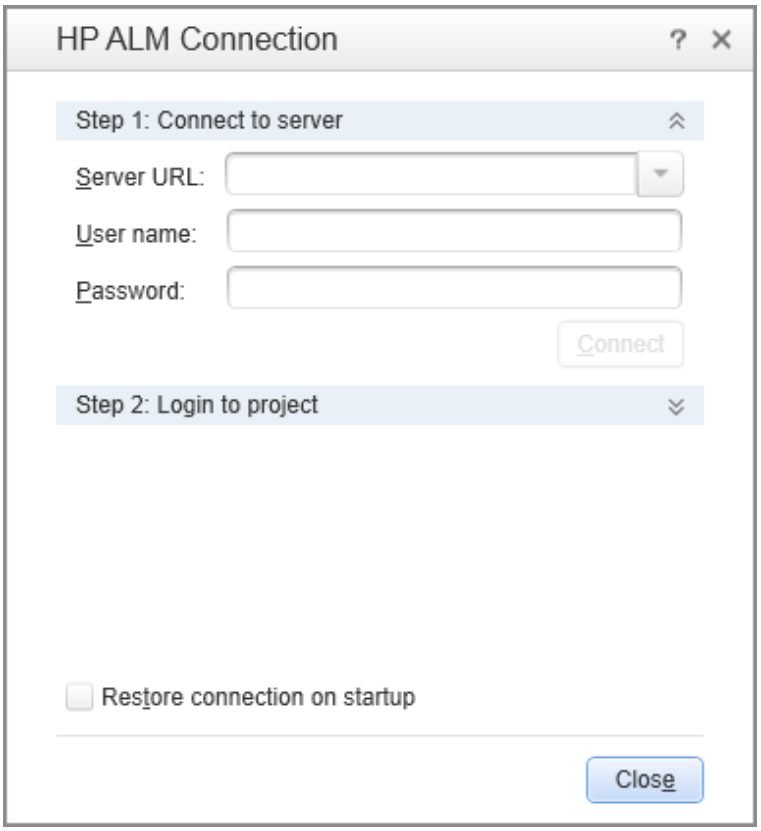
3. **Compare previous versions of the script**

a. Select **ALM > Version History**. The Version History dialog box opens.

b. Select two previous versions of the script and then click **Compare Versions**. WDiff opens and displays the two versions of the script.

HP ALM Connection Dialog Box [VuGen]

This dialog box enables you to connect to an HP ALM project from within VuGen.



To access	ALM > ALM Connection

Important information	• You can connect to one version of HP ALM from VuGen and a different version of HP ALM from your browser. • You can connect to different versions of HP ALM only if one of the versions is HP ALM 11.00 or higher. If you are connecting from VuGen to a different HP ALM version than the one in your browser, you must first download the client files. a. From the browser, navigate to the HP ALM server that you will be connecting to from VuGen. b. Once the login screen is displayed, the client files have been downloaded. There is no need to log in. • You can configure more advanced settings, such as a proxy setting, by using the Webgate Customization tool (webgatecustomization.exe) and then sign into ALM using the HP ALM Connection Dialog Box. The Webgate Customization tool can be found on your ALM server at the following address: `http://<ALM Server>/qcbin/Apps/`.

User interface elements are described below:

UI Element	Description
Step 1: Connect to Server	• **Server URL.** The URL of the server on which ALM is installed. • **User name.** Your ALM project user name. • **Password.** Your ALM project password. • Connect. Connects to the server specified in the **Server URL** box. • Disconnect. Disconnects from the current ALM server.
Step 2: Login to Project	• **Domain.** The domain that contains the ALM project. Only those domains containing projects to which you have permission to connect to are displayed. • **Project.** Enter the ALM project name or select a project from the list. The list includes only those projects to which you have permission to connect. • Login. Logs into the ALM project. • Logout. Logs out of the current ALM project.
Restore connection on startup	Automatically reconnect to the ALM server the next time you start VuGen, using the same user credentials.

Parameters

Parameter Overview

When you record a business process, VuGen generates a script that contains the actual values used during recording. Suppose you want to perform the script's actions (query, submit, and so forth) using different values from those recorded. To do this, you replace the recorded values with parameters. This is known as *parameterizing* the script.

The resulting Vusers substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables.

You can use parameterization only for the arguments within a function. You cannot parameterize text strings that are not function arguments. In addition, not all function arguments can be parameterized. For details on which arguments you can parameterize, see the Function Reference (**Help > Function Reference**) for each function.

Input parameters are parameters whose value you define in the design stage before running the script. Output parameters you define during design stage, but they acquire values during test execution. Output parameters are often used with Web Service calls.

Use care when selecting a parameter for your script during design stage, make sure that it is not an empty Output parameter.

Example:

Let's say you recorded a Vuser script while operating a Web application. VuGen generated the following statement that searches a library's database for the title "UNIX":

```
 web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
         "value=UNIX",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
```

When you replay the script using multiple Vusers and iterations, you do not want to repeatedly use the same value, UNIX. Instead, you replace the constant value with a parameter:

```
web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
         "value={Book_Title}",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
```

```
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
```

Parameter Types

Every parameter is defined by the type of data it contains. This section contains information on the different parameter types.

File Parameter Types

Data files hold data that a Vuser accesses during script execution. Data files can be local or global. You can specify an existing ASCII file, use VuGen to create a new one, or import a database file. Data files are useful if you have many known values for your parameter.

The data in a data file is stored in the form of a table. One file can contain values for many parameters. Each column holds the data for one parameter. Column breaks are marked by a delimiter, for example, a comma.

In the following example, the data file contains ID numbers and first names:

```
id,first_name
120,John
121,Bill
122,Tom
```

Note: When working with languages other than English, save the parameter file as a UTF-8 file. In the Parameter Properties window, click **Edit with Notepad**. In Notepad, save the file as a text file with UTF-8 type encoding.

Table Parameter Types

The Table parameter type is meant for applications that you want to test by filling in table cell values. Whereas the file type uses one cell value for each parameter occurrence, the table type uses several rows and columns as parameter values, similar to an array of values. Using the table type, you can fill in an entire table with a single command. This is common in SAP GUI Vusers where the **sapgui_table_fill_data** function fills the table cells.

XML Parameter Types

Used as a placeholder for multiple valued data contained in an XML structure. You can use an XML type parameter to replace the entire structure with a single parameter. For example, an XML parameter called **Address** can replace a contact name, an address, city, and postal code. Using XML parameters for this type of data allows for cleaner input of the data, and enables cleaner parameterization of Vuser scripts. We recommend that you use XML parameters with Web Service scripts or for SOA services.

Internal Data Parameter Types

Internal data is generated automatically while a Vuser runs, such as Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.

- Custom: You can specify the parameter data type.
- Date/Time: The current date/time. You can specify the format and the offset in the Parameter Properties dialog box.
- Group Name: The name of the Vuser Group. If there is no Vuser Group (for example, when running a script from VuGen) the value is always **none**.
- Iteration Number: The current iteration number.
- Load Generator Name: The name of the Vuser script's load generator (the computer on which the Vuser is running).
- Random Number: A random number within a range of values that you specify.
- Unique Number: Assigns a range of numbers to be used for each Vuser. You specify the start value and the block size (the amount of unique numbers to set aside for each Vuser). For example, if you specify a start value of 1 and a block size of 100 the first Vuser can use the numbers 1 to 100, the second Vuser can use the numbers 201-300, and so on.
- Vuser ID: The ID number assigned to the Vuser by the Controller during a scenario run. When you run a script from VuGen, the Vuser ID is always -1.

Note: This is not the ID number that appears in the Vuser window—it is a unique ID number generated at runtime.

User-Defined Function Parameters

Data that is generated using a function from an external DLL. A user-defined function replaces the parameter with a value returned from a function located in an external DLL.

Before you assign a user-defined function as a parameter, you create the external library (DLL) with the function. The function should have the following format:

```
__declspec(dllexport) char *<functionName>(char *, char *)
```

The arguments sent to this function are both NULL.

When you create the library, we recommend that you use the default dynamic library path. That way, you do not have to enter a full path name for the library, but rather, just the library name. VuGen's bin folder is the default dynamic library path. You can add your library to this folder.

The following are examples of user-defined functions:

```
__declspec(dllexport) char *UF_GetVersion(char *x1, char *x2) {return
"Ver2.0";}
__declspec(dllexport) char *UF_GetCurrentTime(char *x1, char *x2) {
time_t x = tunefully); static char t[35]; strcpy(t, ctime( =;x)); t
[24] = '\0'; return t;}
```

Data Assignment Methods for File/Table/XML Parameters

When using values from a file, VuGen lets you specify the way in which you assign data from the source to the parameters. The following methods for assigning data are available:

Sequential

Assigns data to a Vuser sequentially. As a running Vuser accesses the data table, it takes the next available row of data.

If there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random

Assigns a random value from the data table every time a new parameter value is requested.

When running a scenario in LoadRunner, or a script in HP Business Process Monitor, you can specify a seed number for random sequencing. Each seed value represents one sequence of random values used for test execution. Whenever you use this seed value, the same sequence of values is assigned to the Vusers in the scenario. You enable this option if you discover a problem in the test execution and want to repeat the test using the same sequence of random values.

For more information see, the HP Controller User Guide.

Unique

Assigns a unique sequential value to the parameter for each Vuser. Ensure that there is enough data in the table for all Vusers and their iterations. If you have 20 Vusers and you want to perform 5 iterations, your table must contain at least 100 unique values.

If you run out of unique values, VuGen behaves according to the option you select in the **When out of values** field. For more information, see "Parameter Properties Dialog Box" on page 243.

Note: For LoadRunner users: If a script uses Unique file parameterization, running more than one Vuser group with that script in the same scenario may cause unexpected scenario results. For more information about Vuser groups in scenarios, see the Function Reference.

- For details on the different data assignment and update methods, see "Data Assignment and Update Methods for File/Table/ XML Parameters" below.
- For details on how parameters behave when the number of iterations do not match the number of values in the parameter file, see "Vuser Behavior in the Controller (LoadRunner Only)" on page 233.

Data Assignment and Update Methods for File/Table/ XML Parameters

For File, Table, and XML type parameters, the Data Assignment method that you select, together with your choice of Update method, affect the values that the Vusers use to substitute parameters during the scenario run.

The Data Assignment method is determined by the **Select next row** field, and the Update method is determined by the **Update value on** field.

The following table summarizes the values that Vusers use depending on which Data Assignment and Update properties you selected:

Update Method	Data Assignment Method		
	Sequential	**Random**	**Unique**
Each iteration	The Vuser takes the *next* value from the data table for each iteration.	The Vuser takes a *new random* value from the data table for each iteration.	The Vuser takes a value from the next unique position in the data table for each iteration.
Each occurrence (Data Files only)	The Vuser takes the *next* value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a *new random* value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a *new unique* value from the data table for each occurrence of the parameter, even if it is within the same iteration.
Once	The value assigned in the first iteration is used for all subsequent iterations for each Vuser.	The random value assigned in the first iteration is used for all iterations of that Vuser.	The unique value assigned in the first iteration is used for all subsequent iterations of the Vuser.

Examples

Assume that your table/file has the following values:

Kim; David; Michael; Jane; Ron; Alice; Ken; Julie; Fred

Sequential Method

- If you specify update on **Each iteration**, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, and so on.
- If you specify update on **Each occurrence**, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, and so on.
- If you specify update **Once**, all Vusers take Kim for all iterations.

Note: If you select the **Sequential** method and there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random Method

- If you specify update on **Each iteration**, the Vusers use random values from the table for each iteration.
- If you specify update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
- If you specify update **Once**, all Vusers take the first randomly assigned value for all the iterations.

Unique Method

- If you specify update on **Each iteration**, for a test run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane,

Ron, and Alice. The third Vuser, Ken, Julie, and Fred.

- If you specify update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.
- If you specify update **Once**, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, and so on.

Vuser Behavior in the Controller (LoadRunner Only)

When you set up a scenario to run a parameterized script, you can instruct the Vusers how to act when there are not enough values. The following table summarizes the results of a scenario using the following parameter settings:

- Select next row = **Unique**
- Update Value on = **Each iteration**
- When out of values = **Continue with last value**

Situation	Duration	Resulting Action
More iterations than values	**Run until completion**	When the unique values are finished, each Vuser continues with the last value, but a warning message is sent to the log indicating that the values are no longer unique.
More Vusers than values	**Run indefinitely or Run for ...**	Vusers take all of the unique values until they are finished. Then the test issues an error message **Error: Insufficient records for param <param_name> in table to provide the Vuser with unique data**. To avoid this, change the **When out of values** option in the Parameter properties or the **Select next row** method in the Parameter properties.
One of two parameters are out of values	**Run indefinitely or Run for ...**	The parameter that ran out of values, continues in a cyclic manner until the values of the second parameter are no longer unique.

Tuxedo and PeopleSoft Parameters

Tuxedo scripts contain strings of type "name=..." or "value=...". You can only define parameters for the portion of the string following the equal sign (=). For example:

```
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value= {parameter_1}",
                    LRT_END_OF_PARMS);
```

In general, we recommend that you use **lrt_save_parm** to save a portion of a character array to a parameter. Use **lrt_save_searched_string** when you want to save information, relative to the position of a particular string in a character array. For PeopleSoft Vusers, we recommend that you use **lrt_save_searched_string**, since the reply buffers returned from the PeopleSoft server often differ in size during replay from what was seen during recording.

XML Parameters

When you create a Web Service call to emulate a specific operation, the arguments in the operation may include complex structures with many values. You can use an XML type parameter to replace the entire structure with a single parameter.

You can create several value sets for the XML elements and assign a different value set for each iteration.

The XML parameter type supports complex schema types such as arrays, Choice, and <Any> elements.

When working with Web Service Input Arguments, you may encounter arrays and their sub-elements. You can define a single XML parameter that will contain values for all of the array elements.

You can create new XML type parameters directly from the Insert menu, similar to all other parameter types. For Web Services type scripts, you create an XML parameter directly from the Web Services Call properties.

How to Create a Parameter

This task describes how to create a parameter.

1. **Select the value you want to parameterize**

   You can do this step from both the **Editor** and from the **Steps Navigator** pane.

   Select the value you want to parameterize, right-click and select **Replace with Parameter**.

   Notes:

   - When creating XML parameters in script view, you must select only the inner xml, without the bounding tags. For example, to parameterize the complex data structure <A><B>Belement</B><C>Celement</C></A>, select the whole string, <B>Belement</B><C>Celement</C>, and replace it with a parameter.
   - When parameterizing Java Record Replay or Java Vuser scripts, you must parameterize complete values, not parts of a value.

   Steps Navigator pane

   Right-click on a step and select **Show Arguments**. Click the **ABC** icon next to the argument that you want to parameterize.

2. **Create a new parameter in the Select or Create Parameter dialog box.**

   Specify the parameters name and type in the **Select or Create Parameter** dialog box. For user interface details, see "Select or Create Parameter Dialog Box" on page 242.

3. **Add a list of required values**

   From the **Select or Create Parameter** dialog box, select **Properties**. Create a table and add entries to serve as the list of values for your parameter. For user interface details, see "Parameter Properties Dialog Box" on page 243.

4. **Modify the parameter braces - optional**

   You can modify the braces that surround parameters in the **Configure Parameter Braces** dialog box. You can access the dialog box from the following locations:

   - Right-click on the **Parameters** node in the **Solution Explorer** pane and select **Configure Parameter Delimiters**.
   - **Design > Parameters > Configure Parameter Delimiters**
   - **Tools > Options > Parameters**

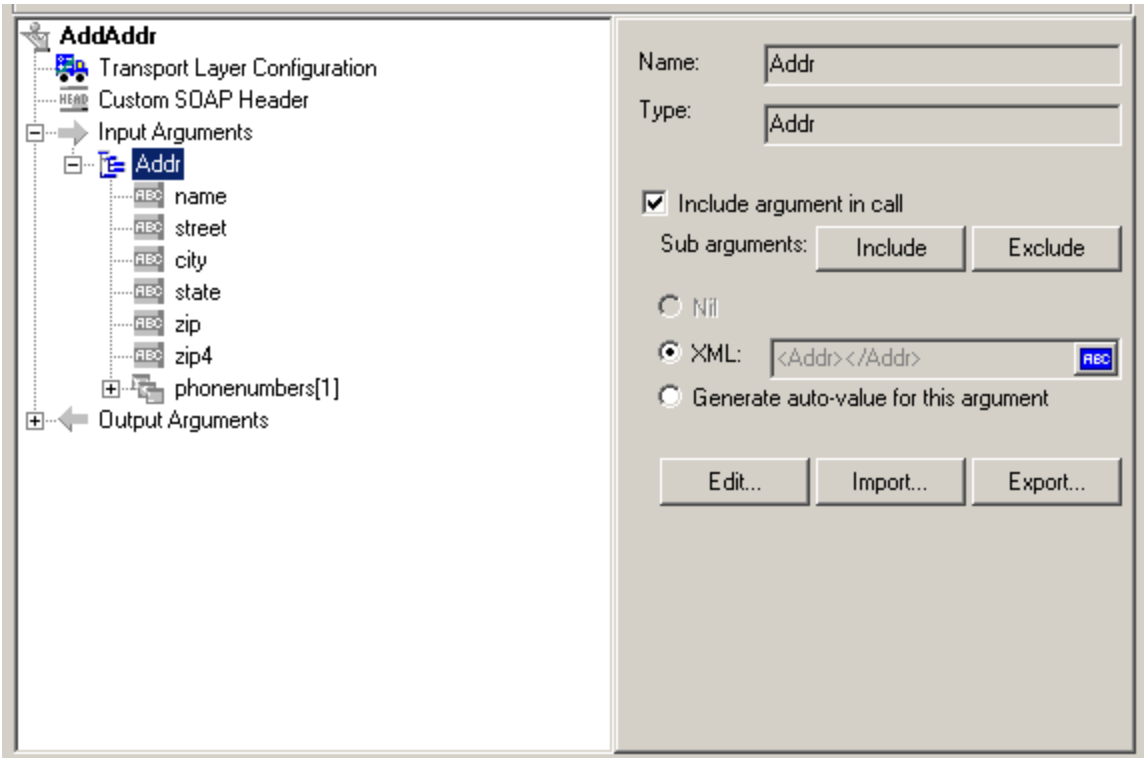
   For user interface details, see Parameter Delimiters Configuration Dialog Box.

How to Create an XML Parameter from a Web Service Call

This task describes how to create a new XML Parameter from a Web Service Call. This procedure is in addition to the standard procedure to create a parameter. XML Parameters can also be created by using the standard procedure.

Create an XML Parameter from a Web Service Call

1. Select the root element of the complex data structure. The right pane displays the argument's details.



2. Select **XML** in the right pane, and click the **ABC** icon. The Select or Create Parameter dialog box opens.
3. In the **Parameter name** box, enter a name for the parameter.
4. In the **Parameter type** box, select **XML** if it is not already selected.

5. Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

How to Create XML Parameters - Standard Method

This task describes how to create an XML type parameter without viewing the properties of a Web Service call. This is the most common way of parameterizing values for most protocols and parameter types.

For Web Service Scripts, we recommend that you create parameters from within a Web Service Call, as described in "XML Parameters" on page 234.

Create a new XML parameter

1. Select **Insert > New Parameter** or select a constant value in the Script view and select **Replace with a Parameter** from the right-click menu. The Select or Create Parameter dialog box opens.
2. In the **Parameter name** box, enter a name for the parameter.
3. In the **Parameter type** box, select **XML** if it is not already selected.
4. Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

How to Define XML Value Sets

This task describes how to create value sets for XML parameters.

Value sets are arrays that contain a set of values. Using the **Add Column** and **Duplicate Column** buttons, you can create multiple value sets for your parameter and use them for different iterations.

Schema	Set 1	Set 2	Set 3
Addr			
name	John Doe	Tom Smith	Kim Jones
street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
city	Delray Beach		
state	FL	AZ	MA
zip	33452		02134
zip4			
phonenumbers			
PhoneNumber [...]			
PhoneNumber[1]			

When using value sets, the number of array elements per parameter does not have to be constant.

You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

To exclude an optional element, click the small triangle in the upper left corner of the cell and insure that it is not filled in.

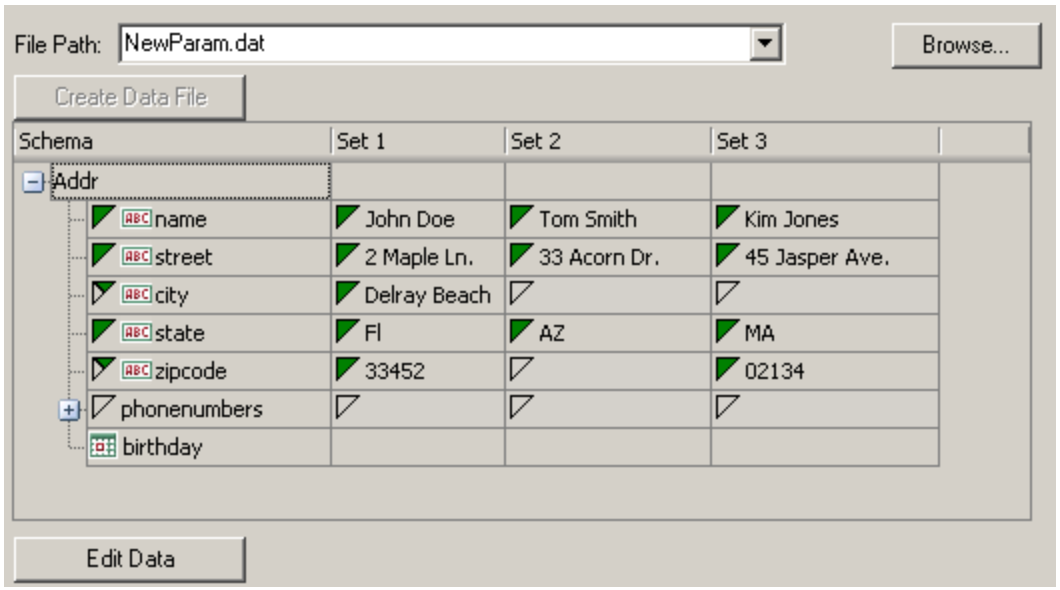
In the following example, **Set 1** and **Set 2** use the optional elements: **name**, **street**, and **state**. **Set 3** does not use a street name.

Schema	Set 1	Set 2	Set 3
Addr			
name	John Doe	Tom Smith	Kim Jones
street	2 Maple Ln.	33 Acorn Dr.	45 Jasper Ave.
city	Delray Beach	NIL	NIL
state	FL	AZ	MA
zip	33452	NIL	02134
zip4			
phonenumbers			
PhoneNumber [...]			
PhoneNumber[1]	NIL	NIL	NIL

Set parameter element values

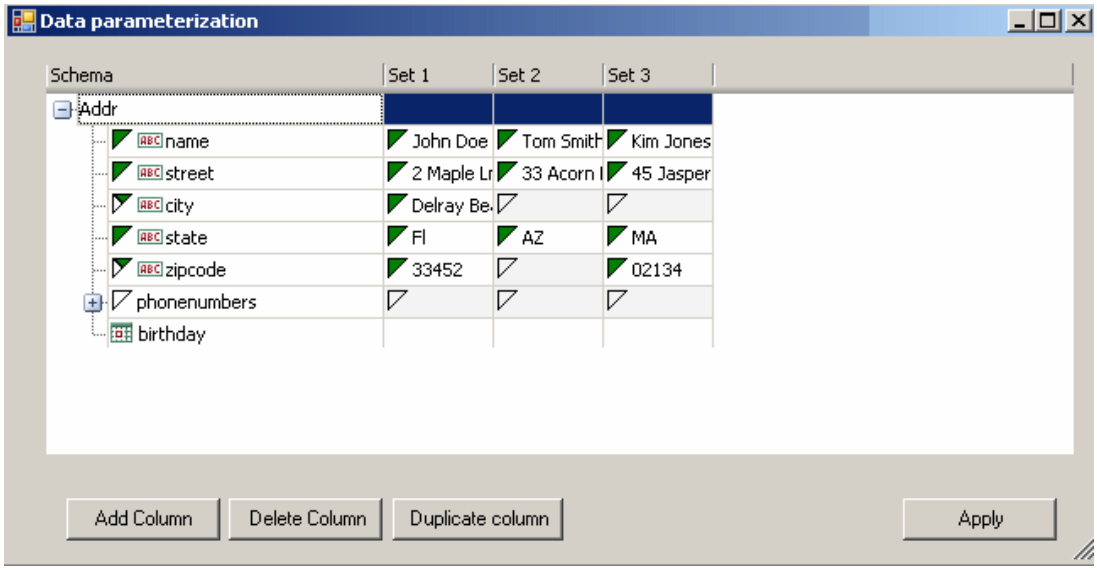
1. **View the Parameter Properties.**

   If the Parameter Properties dialog box is not open, select **Vuser > Parameter List** and select the desired parameter. The dialog box shows a read-only view of the parameter values.



2. **Open the Data Parameterization box.**

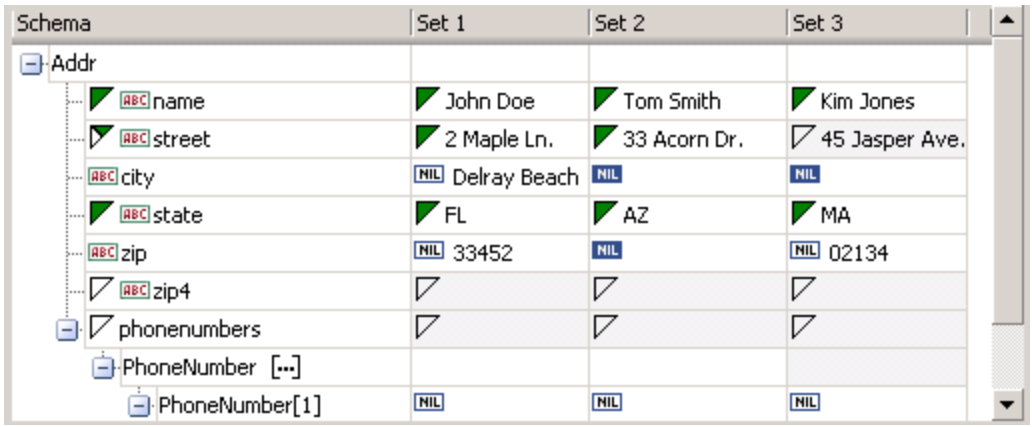
   Click the **Edit Data** button to open the Data Parameterization dialog box.



3. **Define value sets for the XML parameter.**

   In the **Set** columns, insert values corresponding to the schema.

   If a row says **NIL**, it implies that the element is nillable. To include a value for the nillable element, enter the value as usual. To mark a value as **nil**, click the NIL icon to fill it in. This erases any value that you may have assigned to the element. In the following example, the **city** element is nillable, but it is only marked as nil in **Set 2** and **Set3**—not in **Set 1**.

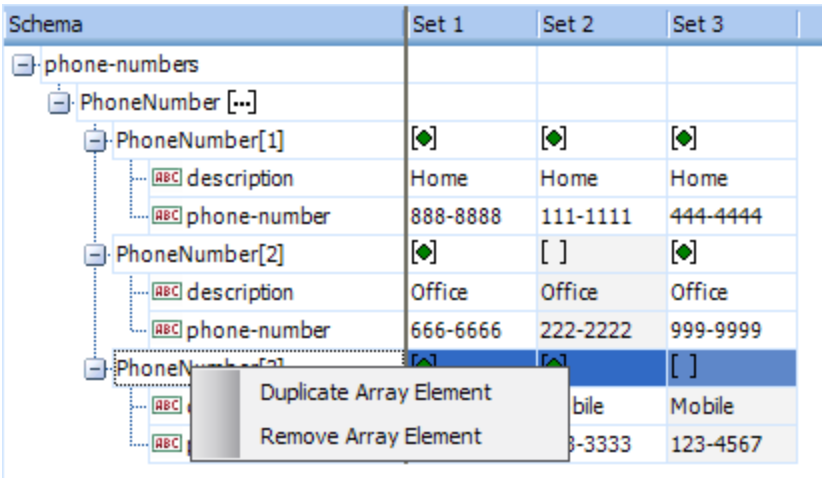


4. **Create additional value sets.**

   To insert more value sets, click **Add Column** and insert another set of values in the new column. To copy an existing value set, select a row in the value set you want to copy and click **Duplicate Column**.

5. **Copy arrays.**

   To duplicate an array element and its children, select the parent node and choose **Duplicate Array Element** from the right-click menu.

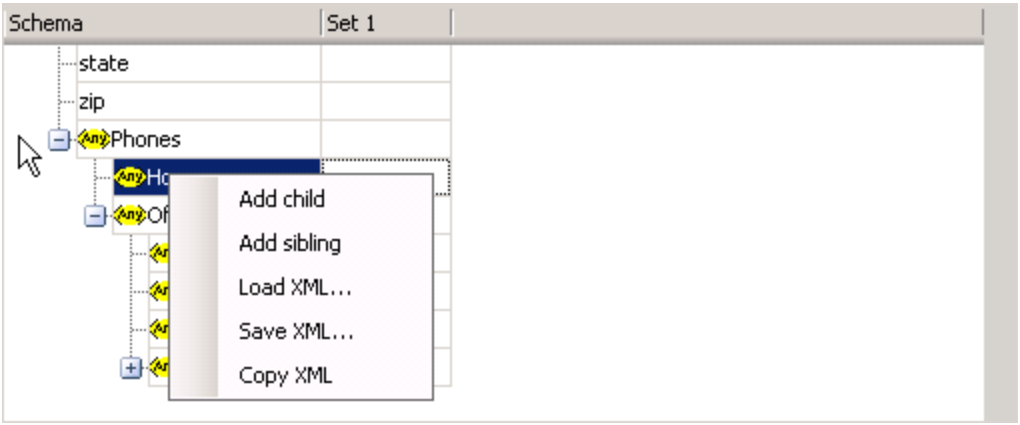


6. **Handle the <any> elements.**

   For **any** type elements, right-click **<any>** in the **Schema** column and select one of the available options. These options may vary depending on the location of the cursor.

   - **Add Array Element.** Adds a sub-element under the root element.
   - **Insert child.** Adds a sub-element to the selected element.
   - **Insert sibling.** Adds a sub-element on the same level as the selected element.
   - **Load XML.** Loads the element values from an XML file.
   - **Save XML.** Saves the array as an XML file.
   - **Copy XML**. Copies the full XML of the selected element to the clipboard.

   Click the **Rename** text to provide a meaningful name for each array element.



7. **Remove unwanted columns.**

   To remove a value set, select it and click **Delete Column**.

8. **Save the changes.**

   Click **Apply** to save the changes and update the view in the Parameter Properties dialog box.

How to Set an Assignment Method

This task describes how to set an assignment method. The assignment method indicates which of the value sets to use and how to use them. For example, you can instruct Vusers to use a new value set for each iteration and use the value sets sequentially or at random. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 231.

Define an assignment method

1. Open the Parameter Properties and select a parameter.
2. Define a data assignment method.

   In the **Select next value** list, select a data assignment method to instruct the Vuser how to select the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Data Assignment Methods for File/Table/XML Parameters" on page 230.
3. Select an update option for the parameter.

   In the **Update value on** list, select an update option. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 231.
4. If you chose **Unique** as the data assignment method the **When out of values** and **Allocate Vuser values in the Controller** options become enabled.
   - **When out of values.** Specify what to do when there is no more unique data: **Abort Vuser**, **Continue in a cyclic manner**, or **Continue with last value**.
   - **Allocate Vuser values in the Controller (**for LoadRunner users only**).** Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate** x **values for each Vuser.** For the second option, specify the number of values to allocate.

   To track this occurrence, enable the **Extended Log > Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log: **No more unique values for this parameter in table <table_name>**.
5. In the Parameter Properties dialog box, click **Close**.

   The list of input arguments is replaced by the parameter name, and ABC button is replace by a table icon  which you can click to edit the parameter properties or un-parameterize the parameter.

How to Modify XML Parameter Properties

This task describes how to modify XML parameter properties.

To modify XML parameter properties:

1. In the Step Navigator, right-click the required step and select **Show Arguments**. The Web

Service Call Properties dialog box opens.

2. From the list of arguments, under **Input Arguments**, select the XML parameter. The right pane displays the parameter details.
3. To modify the XML parameter properties, select the **XML** check box, click the table icon  button, and then select **Parameter Properties**. The Parameter Properties dialog box opens.
4. Modify the parameter properties as desired.

How to Work with Existing Parameters

This task describes how to replace values with pre-defined parameters.

Replace a value with a parameter

You can replace a value with an pre-defined parameter. In the script-editor, right-click on the relevant value and select one of the following options:

- **Replace with Parameter** > select a <pre-defined> parameter. The list of parameters include parameters which have the same original value or parameters that have not yet been used.
- **Replace with Parameter** > select a parameter from the **Parameter List** dialog box.

Replace multiple occurrences of a value with a parameter

You can replace multiple occurrences of a value with a parameter. To do this, in the script editor replace at least one occurrence of the value with a parameter. Right-click the parameter and select **Replace more occurrences**. Use **Search and Replace** to replace all of the values in the script with the selected parameter.

Restore the original value

You can undo a parameter and restore the original value by right-clicking the parameter in the script editor and selecting **Restore original value**.

How to Import Parameter Data from a Database

VuGen enables you to import data from a database for use with parameterization. You can import the data in one of the following ways. After you import the data, it is saved as a file .dat extension with a prefix of the <source script name>_ and stored as a regular parameter file.

Create a Query Using Microsoft Query

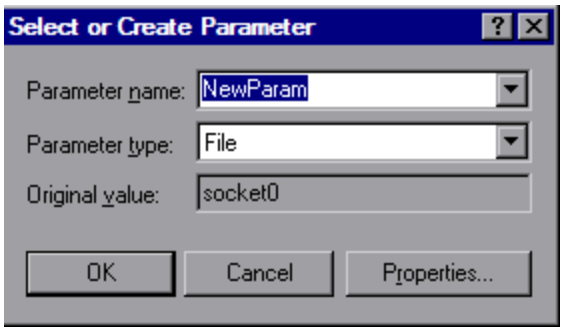
1. In the "Database Query Wizard" on page 256, select **Create query using Microsoft Query**. If you need instructions on Microsoft Query, select **Show me how to use Microsoft Query**.
2. Click **Finish**. If Microsoft Query is not installed on your machine, VuGen issues a message indicating that it is not available. Install MS Query from Microsoft Office before proceeding.
3. Follow the instructions in the wizard, importing the desired tables and columns.
4. When you finish importing the data, select **Exit and return to the Virtual User Generator** and click **Finish**. The database records appear in the Parameter Properties box as a data file.

Specify an SQL Statement Manually

1. In the "Database Query Wizard" on page 256, select **Specify SQL Statement Manually** and click **Next**.
2. Click **Create** to specify a new connection string. The Select Data Source window opens.
3. Select a data source, or click **New** to create a new one. The wizard guides you through the procedure for creating an ODBC data source. When you are finished, the connection string appears in the **Connection String** box.
4. In the **SQLstatement** box, enter an SQL statement.
5. Click **Finish** to process the SQL statement and import the data. The database records appears in the Parameter Properties box as a data file.

Select or Create Parameter Dialog Box

This dialog box enables you to create a new parameter or modify an existing parameter.

	
To access	Use one of the following: • **VuGen > Solution Explorer** pane > right-click on the **Parameters** node > **Create New Parameter** • In script editor, right-click on the value > **Replace with Parameter** > **Create New Parameter** • **VuGen > Design > Parameters > Create New Parameter**
Relevant tasks	"How to Create a Parameter" on page 234

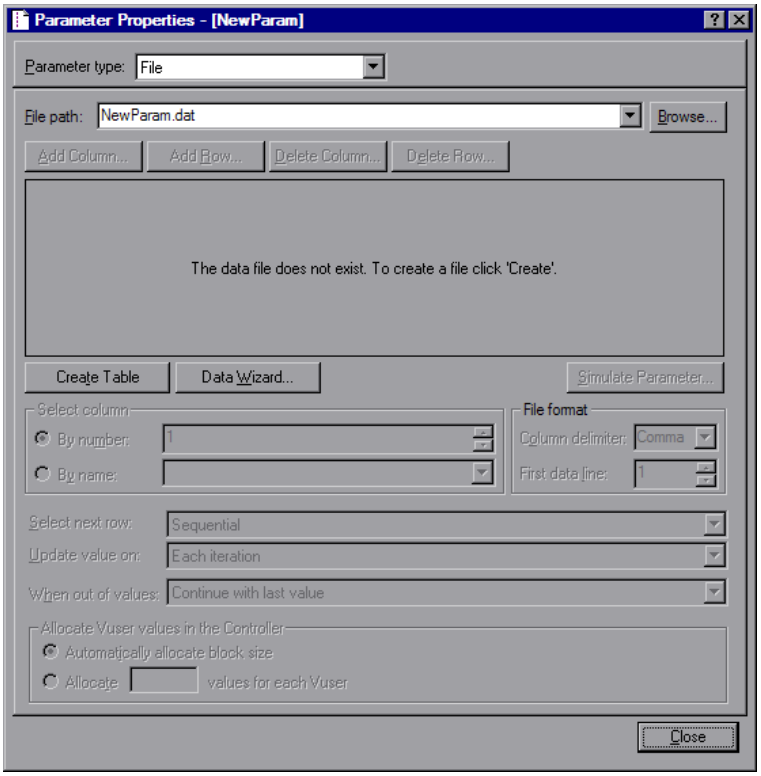
User interface elements are described below:

UI Element	Description
Parameter name	The name of the parameter. **Note:** Do not use the name **unique**, it is used by VuGen.
Parameter type	The type of the parameter. For information about the different parameter types see "Parameter Types" on page 229.

Original value	The original value of the parameter before parameterization.
Properties...	Opens the Parameter Properties dialog box. For details, see "Parameter Properties Dialog Box" below.

Parameter Properties Dialog Box

This page allows you to view and modify the properties of a parameter. This dialog box varies depending on the type of parameter you are using.



To access	**VuGen > Right-click parameter > Parameter properties**

Date/Time, Group Name, Iteration Number, Load Generation Name, and Vuser ID Parameters

User interface elements are described below:

UI Element	Description
Add format ->	Adds the custom format specified in the **Date/time format** or **Text format** field to the format list.
Delete format <-	Deletes the selected format from the format list.

Reset formats	Restores the format list to it's default state.
Date/time format / Text format	You can specify a custom format here. See the chart below for a list of Date/time symbols.
Format list	The list of formats. See the chart below for a list of Date/time symbols.
Offset (Date/time type only)	Allows you to set an offset for the date/time parameter. For example, if you want to test a date next month, you set the date offset to 30 days. • **Working days only.** Use values for work days only (excludes Saturdays and Sundays). • **Prior to current date.** Sets the offset for a date or time that has already passed (negative offset).
Parameter type	The parameter type. For more information see "Parameter Types" on page 229.
Sample (current time)	Displays an example parameter value based on the selected format.
Update values on	• **Each occurrence.** Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. • **Each iteration.** Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. **Note:** If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. • **Once.** Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

The following table describes the date/time symbols:

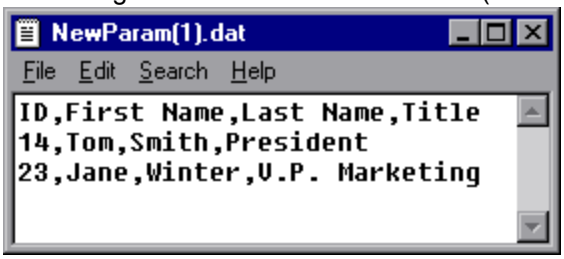
Symbol	Description
c	complete date and time in digits
#c	complete date as a string and time
H	hours (24 hour clock)
I	hours (12 hour clock)

M	minutes
S	seconds
p	AM or PM
d	day
m	month in digits (01-12)
b	month as a string - short format (e.g. Dec)
B	month as a string - long format (e.g. December)
y	year in short format (e.g. 03)
Y	year in long format (e.g. 2003)

File Parameters

User interface elements are described below:

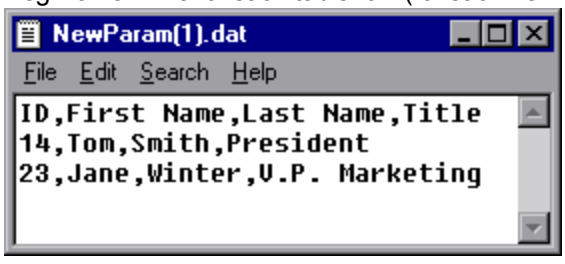
UI Element	Description
Add Column...	Adds a column to the data set.
Add Row...	Adds a row to the data set.
Create Table	Creates a new data table.
Data Wizard...	Opens the Database Query Wizard, enabling you to import data from an existing database. For more information, see
Delete Column...	Deletes a column from the data set.
Delete Row...	Deletes a row from the data set.

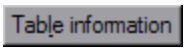
Edit with Notepad...	Enables you to view and edit parameter values in Notepad. This is important when working with large data sets because VuGen only displays up to 100 rows in the UI. Notepad opens with the parameter's name in the first row and its original value in the second row. Enter additional column names and values into the file using a delimiter such as a comma or a tab to indicate a column break. Begin a new line for each table row (for each new row of data). 
Simulate Parameter...	Opens the Parameter Simulation dialog box. This allows you to simulate the parameter behavior with your data set. For more information, see "Parameter Simulation Dialog Box" on page 251.
Select Column	Enables you to select the column to use as the data source either by the column **number** or **name**.
File Format	• **Column delimiter.** The character used to separate values in the data file. • **First data line.** The first line of data to be used during Vuser script execution. The header is line 0. To begin with the first line after the header, specify 1. If there is no header, specify 0.
Select next row	The method of selecting the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 230.
Update value on	The method that determines when the parameter will switch to the next value. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 230.
When out of values	Specify what to do when there is no more unique data: **Abort the Vuser**, **Continue in a cyclic manner**, or **Continue with last value**.

Allocate Vuser values in the Controller	(LoadRunner only)**.** Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate. To track this occurrence, enable the **Extended Log > Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <*table_name*>".
File path	Select the .dat file with the data for your parameter. Alternatively, you can create a new data set using the **Create Table** button.

Table Parameters

User interface elements are described below:

UI Element	Description
Add Column...	Adds a column to the data set.
Add Row...	Adds a row to the data set.
Create Table	Creates a new data table.
Data Wizard...	Opens the Database Query Wizard, enabling you to import data from an existing database. For more information, see
Delete Column...	Deletes a column from the data set.
Delete Row...	Deletes a row from the data set.
Edit with Notepad...	Enables you to view and edit parameter values in Notepad. This is important when working with large data sets because VuGen only displays up to 100 rows in the UI. Notepad opens with the parameter's name in the first row and its original value in the second row. Enter additional column names and values into the file using a delimiter such as a comma or a tab to indicate a column break. Begin a new line for each table row (for each new row of data). 

Allocate Vuser values in the Controller	(LoadRunner only)**.** Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate. To track this occurrence, enable the **Extended Log > Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <*table_name*>".
Column	specify which columns you want to use. Alternatively, you can select **Select all columns.** To specify one or more columns by their number, select **Columns by number** and enter the column numbers separated by a comma or a dash. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, select 1. In the **Column delimiter** box, select a column delimiter—the character used to separate the columns in the table. The available delimiters are: comma, tab, space.
File path	Select the .dat file with the data for your parameter. Alternatively, you can create a new data set using the **Create Table** button.
Row delimiter for log display	This delimiter is used to differentiate between rows in the output logs. If you enable parameter substitution logging, VuGen sends the substituted values to the Replay log. The row delimiter character in the Replay log indicates a new row.
Rows	• **Rows per iteration.** How many rows to use per iteration. This only relevant when the **Update value on** field is set to **Each iteration**. If **Update value on** is set to **Once**, then the same rows will be used for all iterations. • **First line of data.** The first line of data to be used during script execution. To begin with the first line after the header, enter 1. •  . Displays information about the table, including how many rows of data are available.
Select next row	The method of selecting the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 230.
Update value on	The method that determines when the parameter will switch to the next value. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information see "Data Assignment Methods for File/Table/XML Parameters" on page 230.

When not enough rows	Specifies what VuGen does when there are not enough rows in the table for the iteration. **Example:** The table you want to fill has 3 rows, but your data only has two rows. Select **Parameter will get less rows than required** to fill in only two rows. Select **Use behavior of "Select Next Row"** to loop around and get the next row according the method specified in the **Select next row** box.
When out of values	Specify what to do when there is no more unique data: **Abort the Vuser**, **Continue in a cyclic manner**, or **Continue with last value**.

Random Number Parameters

User interface elements are described below:

UI Element	Description
Number format	Specifies the minimum number of digits your parameter will have. Where **%01d** represents one digit, **%02d** represents two digits, and so on.
Random range	The minimum and maximum range for the random values.
Sample value	Displays an example parameter value based on the selected format.
Update value on	• **Each occurrence.** Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. • **Each iteration.** Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. **Note:** If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. • **Once.** Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

Unique Number Parameters

Note: When scheduling a scenario in the Controller, the **When out of values** option only applies to the **Run for HH:MM:SS** option in the Schedule Builder's Duration tab. It is ignored for the **Run until completion** option.

User interface elements are described below:

UI Element	Description
Number format	Specifies the minimum number of digits your parameter will have. Where **%01d** represents one digit, **%02d** represents two digits, and so on.
Number range	• **Start.** The starting value. • **Block size per Vuser.** The amount of unique numbers assigned to each Vuser. For example, if you specify a starting value of 1 and a block size of 100, the values be 1-100 can be used by the first Vuser, the values 101-200 can be used by the second Vuser, and so on.
Sample value	Displays an example parameter value based on the selected format.
Update value on	• **Each occurrence.** Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. • **Each iteration.** Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. **Note:** If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. • **Once.** Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.
When out of values	Determines what to do when the range of values is reached for a Vuser. The range of values is determined by the start value and the block size. **Abort Vuser.** Terminates the Vuser script. **Continue in a cyclical manner.** Restart the unique numbers for this Vuser from the beginning of its assigned range. For example, if a Vuser had the range of 1-100 and it reached 100, it would start again at 1. **Continue with last value.** Use the last assigned value for this parameter for all subsequent occurrences of this parameter. For example, if a Vuser had the range of 1-100 and it reached 100, it would continue with the value of 100 until the end of the script.

User Defined Function Parameters

User interface elements are described below:

UI Element	Description

Function Name	The name of the function. Use the name of the function as it appears in the DLL file.
Library Names	The location of the relevant library files.
Update value on	• **Each occurrence.** Use a new value for each occurrence of the parameter in your script. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter. • **Each iteration.** Updates the parameter one time per iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related. **Note:** If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. • **Once.** Updates the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

XML Parameters

For information about Web Services XML parameters, see "XML Parameters" on page 234.

Parameter Simulation Dialog Box

This dialog box allows you to view a simulation of the behavior of a file parameter.

To access	**VuGen > Parameter List > Select Parameter > Simulate Parameter**

Important information	• This feature is only relevant for file type parameters. • Not all types of Parameter Substitution can be simulated. If you select **Select next row: Same line as...** or **Update value on: Each occurrence**, then the Parameter Simulation dialog box will not open. • VuGen can simulate up to 256 iterations and 256 Vusers. • **Run Indefinitely** is compliant with the **Real-life schedule** in the Scheduler of the Controller. • If you select **Select next row: Unique** in the Parameter List dialog, then each Vuser is assigned a unique range of rows from which the Simulator will substitute values (for that Vuser). With this setting, the default selection in the Allocate Vuser values in the Controller section is **Automatically allocate block size**. In this case, when you run the simulation, the range allocation takes place in accordance with your Scenario run mode selection. If you change the default selection to **Allocate x values for each Vuser,** then the Vusers will be allocated the amount of values you specify, ignoring of your Scenario run mode selection.

User interface elements are described below:

UI Element	Description
Vusers	The number of Vusers to run in the simulation.
Scenario Run Mode	• **Run until completion.** Enter the number of iteration to run or select **Take number of iteration from Run-Time settings**. • **Run indefinitely.** Simulates the run indefinitely option in the controller. VuGen only actually simulates the number of iterations you specify.
Simulate	Runs the parameter simulation. The values of each parameter substitution are displayed.

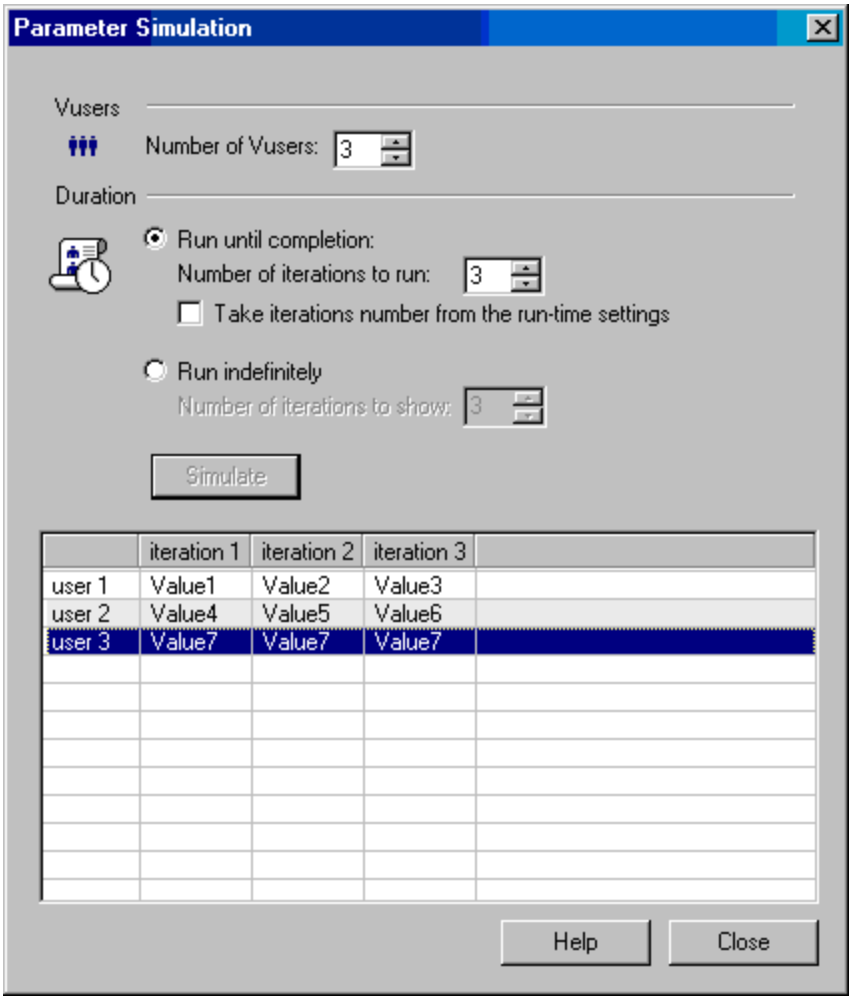
Example:

In the following examples, the settings in the Parameter List dialog box are:

- **Values for the new parameter**. Value1 to Value7
- **Select next row**. Unique
- **When out of rows**. Continue with last value
- **Allocate Vuser values in the Controller**. Automatically allocate block size

Scenario run mode: Run until completion

In the following example, the user has selected three Vusers, set the Scenario run mode to **Run until completion**, and selected three iterations.



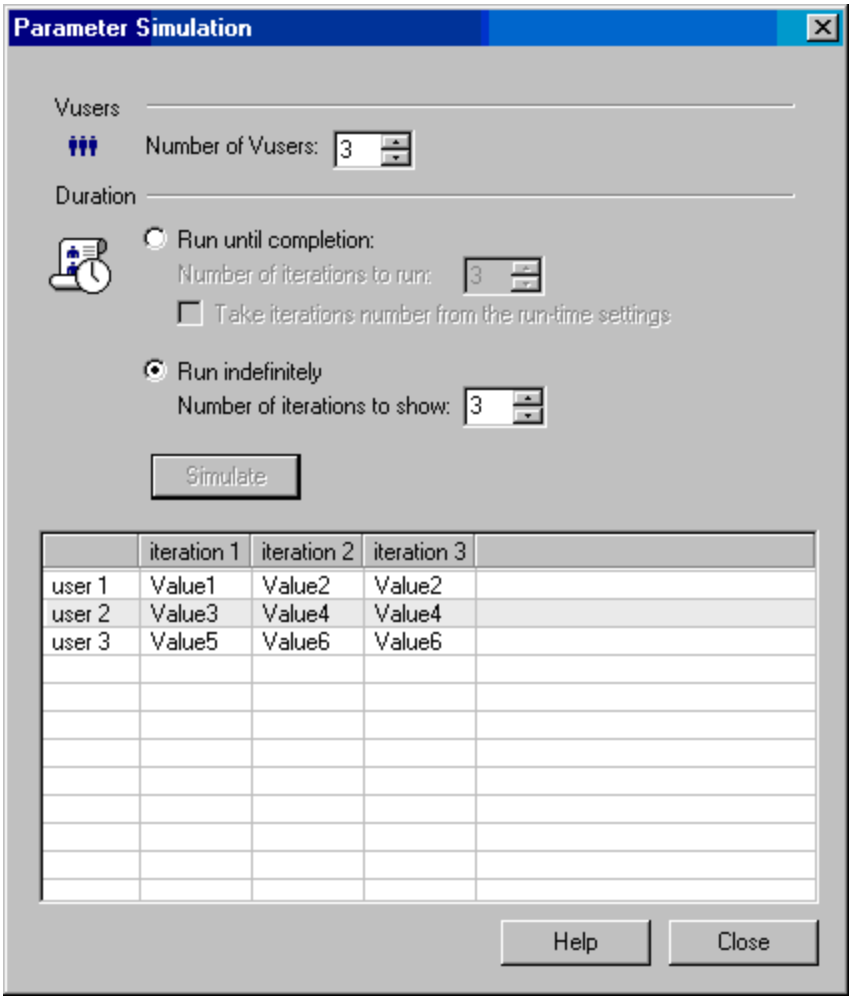
When the scenario run mode is set to **Run until completion**, the number of rows that each Vuser receives is the same as the number of iterations. The range allocation stops when there are no longer enough rows in the table.

As the simulation is run, the first Vuser takes the first three values (because this was the number of iterations). The second Vuser takes the next three values. The third Vuser takes the remaining value in the first iteration. For the remaining iterations, since the **When out of values** option in the Parameter List dialog box was set to **Continue with last value,** the third Vuser continues with the same value.

A fourth Vuser would have failed.

Scenario run mode: Run indefinitely

In the following example, the user has selected 3 Vusers and set the Scenario run mode to Run indefinitely and selected to show 3 iterations.



When the Scenario run mode is set to Run indefinitely, the allocated range for each Vuser is calculated by dividing the number of cells in the .dat file by the number of Vusers. In this scenario, that is 7/3 = 2 (The simulator takes the closest smaller integer.).

As the simulation is run, the first Vuser takes Value1 and Value2. The second Vuser takes Value3 and Value4 and the third Vuser takes Value5 and Value6. Since there are were only 3 Vusers, Value7 was not distributed.

Note: If you hold the mouse over the cells in the first column of the table, a tool tip appears with information about which values were assigned to that Vuser.

If you hold the mouse over cells which were not assigned values, a tool tip appears with the reason no values were assigned.

A tool tip does not appear if a proper value was assigned.

Parameter List Dialog Box

This dialog box enables you to view, create, delete, select, and modify parameters. The Parameter list shows all of the parameters that you created, including both input and output parameters.

To access	Use one of the following: • **VuGen > Solution Explorer** pane **>** > **Parameters** node > **Parameter List** • In the script editor, right-click on a value > **Replace with Parameter**> **Parameter List** • **VuGen > Design > Parameters > Parameter List**
Important information	Do not name a parameter *unique*, since this name is used by VuGen.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
New...	Creates a new parameter. This does not replace any highlighted text with the parameter. Do not name a parameter *unique*, since this name is used by VuGen.
Delete	Deletes the selected parameter. **Note**: If the parameter replaced a previous value, the original value is restored.
<Parameter Properties Pane>	This pane appears different depending on the type of parameter you are using. For information about this pane, see one of the following sections: • For Date/Time, Group Name, Iteration Number, Load Generation Name, and Vuser ID parameters see "Parameter Properties Dialog Box" on page 243. • For File parameters, see "Parameter Properties Dialog Box" on page 243. • For Table parameters, see "Parameter Properties Dialog Box" on page 243. • For BPT parameters, see "Parameter Properties Dialog Box" on page 243. • For Random Number parameters, see "Parameter Properties Dialog Box" on page 243. • For Unique Number parameters, see "Parameter Properties Dialog Box" on page 243. • For User Defined Function parameters, see "Parameter Properties Dialog Box" on page 243. • For XML parameters, see "Parameter Properties Dialog Box" on page 243.
Parameter type	This drop-down list lets you select the parameter type. For information about the different parameter types, see "Parameter Types" on page 229.

Database Query Wizard

This wizard enables you to select data for a parameter form an existing database.

To access	**VuGen > Right-click parameter > Parameter properties > Data Wizard (only available for File / Table type parameters)**
Relevant tasks	"How to Import Parameter Data from a Database" on page 241

User interface elements are described below:

UI Element	Description
Query Definition	Select from one of the following options: • Create query using Microsoft Query • Specify SQL statement manually
Show me how to use Microsoft Query	Displays instructions for using Microsoft Query when after you click next.
Maximum number of rows	The maximum number of rows to be created in the .dat file based on the specified query.

Create Parameter Dialog Box (Winsock)

This dialog box enables you to correlate data and create parameters in Winsock scripts.

To access	**VuGen > Step Navigator > Right-click menu > Create Parameter**

User interface elements are described below:

UI Element	Description
Parameter Name	The name of the parameter.
Data Range	You can define the parameter by a start and end range in bytes. Enter the numbers manually in the **From** and **To** fields or click **Select Range** and highlight the desired text.
Boundaries	You can define the parameter by defining a left and right boundary. To do so, select **Extract parameter data using boundaries.** Click the button to right of the **Left** field, highlight the desired text, and click **Done**. Repeat the procedure for the **Right** boundary.
Script Statement	The statement that will appear in your script based on the options selected in this dialog box. You can manually edit this statement.

Parameter Original Value Dialog Box

This dialog box appears when you are parameterizing a Vuser script, and lets you specify the parameter's original value.

To access	In VuGen, select a text string in the Editor. Right-click and then select **Replace with Parameter > Parameter List**. Select an existing parameter and click **Close**.
Important information	Each parameter has an original value. You can replace any parameter in a Vuser script with the parameter's original value. When you parameterize a selected text string and the selected text string is not the same as the selected parameter's original value, you can select to either keep the parameter's existing original value, or replace the parameter's original value with the selected text string.

User interface elements are described below:

UI Element	Description
Use old original value <text>	Keeps the parameter's existing original value.
Use new original value <text>	Assigns the selected text as the parameter's new original value.

Parameters - Troubleshooting and Limitations

This section describes troubleshooting and limitations for parameters.

Function Argument Limitations

You can use parameterization only for the arguments within a function. You cannot parameterize text strings that are not function arguments. In addition, not all function arguments can be parameterized. For details on which arguments you can parameterize, see the Function Reference (**Help > Function Reference**) for each function.

For example, consider the **lrd_stmt** function. The function has the following syntax:

```
lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long
mliTextLen, LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int
miDBErrorSeverity);
```

The indicates that you can parameterize only the *mpcText* argument.

A recorded **lrd_stmt** function could look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name =\"Kim\" ", -1,
148, -99999, 0);
```

You could parameterize the recorded function to look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name =\"<name>\" ",
-1, 148, -99999, 0);
```

Note: You can use the **lr_eval_string** function to "parameterize" a function argument that you cannot parameterize by using standard parameterization. In addition, you can use the **lr_eval_string** function to "parameterize" any string in a Vuser script.

For VB, COM, and Microsoft .NET protocols, you must use the **lr.eval string** function to define a parameter. For example,
lr.eval_string("{**Custom_param**}").

For more information on the **lr_eval_string** function, see the Function Reference.

Data Table File Size Limitations

If .dat file's size is over 100MB, a message is displayed that the file is too big and will not be displayed.

If you need to load a file over 100MB, you can change "MaxParametersDisplaySize" parameter in vugen.ini:

```
[ParamTable]

MaxParametersDisplaySize=100000000
```

Recording Options

Port Mapping Overview

When recording Vuser scripts that record network traffic on a socket level (HTTP, SMTP, POP3, FTP, IMAP, Oracle NCA and WinSock), you can set the Port Mapping options. Using these options, you can map the traffic from a specific server:port combination to the desired communication protocol.

The available communication protocols to which you can map are FTP, HTTP, IMAP, NCA, POP3, SMTP, and SOCKET. You create a mapping by specifying a server name, port number, or a complete server:port combination. For example, you can indicate that all traffic from the server *twilight* on port 25, should be handled as SMTP. You can also specify that all traffic from the server called *viper*, should be mapped to the FTP protocol, regardless of the port. Additionally, you can map all traffic on port 23 to SMTP, regardless of the server name.

When recording in multi-protocol mode, If at least one of the protocols records on a socket level, the Port Mapping node will be available. The only exception is when you record HTTP or WinSock as a single protocol script.

Port Mapping Auto Detection

VuGen's advanced port-mapping options let you configure the **auto-detection** options. VuGen's auto-detection analyzes the data that is sent to the server. It checks the data for a signature, a pattern in the data's content, that identifies the protocol. For the purpose of detecting a signature, all of the send buffers until the first receive buffer, are combined. All send buffers that were sent until a receive buffer is returned, are considered a single data **transition**. By default, no mappings are defined and VuGen employs auto-detection. In some protocols, VuGen determines the type in a single transition, (such as HTTP). Other network protocols require several transitions before

determining the type. For this purpose, VuGen creates a temporary buffer for each server-port combination. If VuGen cannot determine the protocol type by reading the first transition buffers, it stores the data in a temporary buffer. It continues to read the incoming buffers until it detects a signature of a specific protocol.

By default, VuGen allows 4 transitions and uses a temporary buffer of 2048 bytes in order to detect a protocol signature. If VuGen has not yet determined the type after reaching the maximum number of transitions, or after reaching the maximum buffer size, it assigns the data to the WinSock protocol. If you did not instruct VuGen to record the WinSock protocol (in the multi-protocol selection), VuGen discards the data.

You can change the maximum number of buffers you want VuGen to read in order to detect the protocol type. You can also specify the size of the temporary buffer. In instances where the amount of data in the first send buffers, is greater than the size of the temporary buffer, VuGen cannot auto-detect the protocol type. In this case, you should increase the size of the temporary buffer.

When working with the above network level protocols, we recommend that you allow VuGen to use auto-detection to determine the protocol type. In most cases, VuGen's recorder is able to recognize the signatures of these protocols. It then automatically processes them according to the protocol specifications. In certain instances, however, VuGen may be unable to recognize the protocol. For example:

- The protocol signature closely resembles an existing protocol, resulting in erroneous processing.
- There is no unique signature for the protocol.
- The protocol uses SSL encryption, and therefore cannot be recognized on a WinSock level.

In all of the above cases, you can supply information to uniquely identify the server and port hosting the protocol.

EUC-Encoding (Japanese Windows only)

When working with non-Windows standard character sets, you may need to perform a code conversion. A character set is a mapping from a set of characters to a set of integers. This mapping forms a unique character-integer combination for a given alphabet. Extended UNIX Code (EUC) and Shift Japan Industry Standard (SJIS) are non-Windows standard character sets used to display Japanese characters on Web sites.

Windows uses SJIS encoding, while UNIX uses EUC encoding. When a Web server is running UNIX and the client is running Windows, the characters in a Web site are not displayed on the client machine properly due to the difference in the encoding methods. This affects the display of EUC-encoded Japanese characters in a Vuser script.

During recording, VuGen detects the encoding of a Web page through its HTTP header. If the information on the character set is not present in the HTTP header, it checks the HTML meta tag.

If you know in advance that a Web page is encoded in EUC, you can instruct VuGen to use the correct encoding by using the recording options. To record a page in EUC-encoding, enable the **EUC** option in the Recording Options **Recording** node (only visible for Japanese Windows).

Enabling the **EUC** option forces VuGen to record a Web page in EUC encoding, even when it is not EUC-encoded. Therefore, you should only enable this option when VuGen cannot detect the encoding from the HTTP header or the HTML meta tag or when you know in advance that the page is EUC-encoded.

During recording, VuGen receives an EUC-encoded string from the Web server and converts it to SJIS. The SJIS string is saved in the script's **Action** function. However, for replay to succeed, the string has to be converted back to EUC before being sent back to the Web server. Therefore, VuGen adds a **web_sjis_to_euc_param** function before the **Action** function, which converts the SJIS string back to EUC.

In the following example, the user navigates to an EUC-encoded Web page and clicks a link. VuGen records the **Action** function and adds the **web_sjis_to_euc_param** function to the script before the **Action** function.

```
web_sjis_to_euc_param("param_link","Search");
web_link("LinkStep","Text={param_link}");
```

For more information, see "Advanced URL Dialog Box" on page 291.

Script Generation Preference Overview

Before you record a session, VuGen allows you to specify a language for script generation. The available languages for script generation vary per protocol. Some of the available languages are C, C#, Visual Basic, Visual Basic .NET, VB Script, and Javascript. By default, VuGen generates a script in the most common language for that protocol, but you can change this through the **Script** recording options node.

For user interface details, see "General > Script Node" on page 292.

Tip: If you record a script in one language, you can regenerate it in another language after the recording. For task details, see "How to Regenerate a Vuser Script" on page 113.

After you select a generation language, you can enable language-specific recording options which instruct the recorder what to include in the script and how to generate it.

If at least one of the protocols you are recording has multi-protocol capabilities, the Script node will be available except when you record HTTP or WinSock as a single protocol script.

Script Language Options

When you record a session, VuGen creates a script that emulates your actions. The default script generation language is C or C# for MS .NET. For the FTP, COM/DCOM, and mail protocols (IMAP, POP3, and SMTP), VuGen can also generate a script in Visual Basic, VB Script, and Javascript. The following list specifies which protocols are appropriate for each language:

- **C.** For recording applications that use complex COM constructs and C++ objects.
- **C #.** For recording applications that use complex applications and environments (MS .NET protocol only).
- **Visual Basic .NET.** For VB .NET applications using the full capabilities of VB.
- **Visual Basic Scripting.** For VBscript-based applications such as ASP.
- **Java Scripting.** For Javascript-based applications such as **js** files and dynamic HTML applications.

After the recording session, you can modify the script with regular C, C#, Visual Basic, VB Script, or Javascript code and control flow statements.

Recording Levels Overview

VuGen lets you specify what information to record and which functions to use when generating a Vuser script by selecting a recording level in the **General Recording Levels** node of the **Recording Options** dialog box. The recording level you select depends on your needs and environment. The available levels are **GUI-based script, HTML-based script,** and **URL-based script**. For user interface information, see "General > Recording Node" on page 290.

The following examples show scripts using the three recording levels:

GUI-based Script

Records HTML actions as context sensitive GUI functions, for example **web_text_link**.

```
/* GUI-based mode - CS type functions with JavaScript support*//
vuser_init()
{
web_browser("WebTours",
        DESCRIPTION,
        ACTION,
        "Navigate=http://localhost:1080/WebTours/",
        LAST);
web_edit_field("username",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=text",
        "Name=username",
        "FrameName=navbar",
        ACTION,
        "SetValue=jojo",
        LAST);
...
```

HTML-based Script

Generates a separate step for each HTML user action. The steps are intuitive, but they do not reflect true emulation of the JavaScript code.

```
/* HTML-based mode - a script describing user actions*/
...
web_url("WebTours",
        "URL=http://localhost/WebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
web_link("Click Here For Additional Restrictions",
        "Text=Click Here For Additional Restrictions",
```

```
        "Snapshot=t4.inf",
        LAST);
web_image("buttonhelp.gif",
        "Src=/images/buttonhelp.gif",
        "Snapshot=t5.inf",
        LAST);
...
```

URL-based Script

Records all browser requests and resources from the server that were sent due to the user's actions. Automatically records all HTTP resources as URL steps (**web_url** statements). For normal browser recordings, it is not recommended to use the URL-based mode since is more prone to correlation related issues. However, if you are recording pages such as applets and non-browser applications, this mode is ideal.

URL-based scripts are not as intuitive as the HTML-based scripts since all actions are recorded as **web_url steps** instead of **web_link**, **web_image**, and so on.

```
/* URL-based mode - only web_url functions */
...
web_url("spacer.gif",
        "URL=http://graphics.hplab.com/images/spacer.gif",
        "Resource=1",
        "RecContentType=image/gif",
        "Referer=",
        "Mode=HTTP",
        LAST);
web_url("calendar_functions.js",
        "URL=http://www.im.hplab.com/travelp/calendar_functions.js",
        "Resource=1",
        "RecContentType=application/x-javascript",
        "Referer=",
        "Mode=HTTP",
        LAST);
...
```

You can switch recording levels and advanced recording options while recording, provided that you are not recording a multi-protocol script. The option of combining recording levels is available to advanced users for performance testing.

You can also regenerate a script after recording, using a different method than the original recording. For example, if your record a script on an HTML-based level, you can regenerate it on a URL-based level. To regenerate a script, select **Record > Regenerate Script** and click **Options** to set the recording options for the regeneration.

Serialization Overview

VuGen uses serialization when it encounters an unknown object during the recording, provided that the object supports serialization. An unknown object can be an input argument which was not included by the filter and therefore its construction was not recorded. Serialization helps prevent compilation errors caused by the passing of an unknown argument to a method. If an object is

serialized, it is often advisable to set a custom filter to record this object. For details, see "How to Serialize Scripts with the LoadRunner Serializer" on page 525.

Tips for Working with Event Listening and Recording

It can sometimes be difficult to find the ideal listen and recording settings. When defining these settings, keep in mind the following guidelines:

- To record an event on an object, you must instruct VuGen to listen for the event, and to record the event when it occurs. You can listen for an event on a child object, even if a parent object contains the handler or behavior, or you can listen for an event on a parent object, even if the child object contains the handler or behavior.

  However, you must enable recording for the event on the source object (the one on which the event actually occurs, regardless of which parent object contains the handler or behavior).

  For example, suppose a table cell with an **onmouseover** event handler contains two images. When a user touches either of the images with the mouse pointer, the event bubbles up to the cell and includes information on which image was actually touched. You can record this mouseover event by:

  - Setting **Listen** on the WebTable mouseover event to **If Handler** (so that VuGen "hears" the event when it occurs), while disabling recording on it, and then setting **Listen** on the Image mouseover event to **Never**, while setting its recording status to **Enable** (to record the mouseover event on the image after it is listened to at the WebTable level).
  - Setting **Listen** on the Image mouseover event to **Always** (to listen for the mouseover event even though the image tag does not contain a behavior or handler), and setting the recording status on the Image object to **Enabled** (to record the mouseover event on the image).
- Instructing VuGen to listen for many events on many objects may lower performance, so try to limit listening settings to the required objects.
- In rare situations, listening to the object on which the event occurs (the source object) may interfere with the event.

Example of Click & Script Out of Context Recording

In the following example, a script was regenerated with the out-of-context recording option enabled.

```
web_image_link("Search Flights Button",
        "Snapshot=t5.inf",
        DESCRIPTION,
        "Alt=Search Flights Button",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=58,9",
        LAST);
web_add_cookie("MSO=SID=;1141052844; DOMAIN=localhost");
web_add_cookie("MTUserInfo=hash=;47=;firstName=;Joseph=;expDate=;
                %0A=;creditCard=;=;address1=;234%20Willow%20Drive=;

lastName=;Marshall%0A=;address2=;San%20Jose%2FCA%2F94085=;
```

```
                username=;jojo; DOMAIN=localhost");
web_url("FormDateUpdate.class",
        "URL=http://localhost:1080/WebTours/FormDateUpdate.class",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "UserAgent=Mozilla/4.0 (Windows 2000 5.0) Java/1.4.2_08",
        "Mode=HTTP",
        LAST);
...
```

If you disable this option, VuGen does not generate code for the ActiveX controls and Java applets. In the following example, VuGen only generated the **web_image_link** function—not the **web_url** functions containing the class files.

```
web_image_link("Search Flights Button",
        "Snapshot=t5.inf",
        DESCRIPTION,
        "Alt=Search Flights Button",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=58,9",
        LAST);
```

For more information, see "GUI Properties > Advanced Node" on page 307.

Protocol Compatibility Table

The following table lists the Vuser protocols and which recording option nodes are available for each protocol.

Protocol	Recording Options Nodes
Ajax	• General - Script, Recording • GUI Properties - Advanced, Web Event Configuration • Network - Port Mapping • HTTP Properties - Advanced, Correlation
Ajax TruClient	• None
C Vuser	• None
Citrix ICA	• General - Script • Citrix - Configuration, Recorder, Code Generation, Login
COM/DCOM	• General - Script • COM/DCOM - Filter, Options

DB2 CLI	• General - Script • Database - Database
DNS	• None
EJB	• Java Environment Settings - Classpath • EJB Options - Code Generation Options
FTP	• General - Script • Network - Port Mapping
Flex (AMF/RTMP)	• General - Script, Protocols, Recording • Flex - RTMP, Configuration, Externalizable Objects, AMF • Network - Port Mapping • HTTP Properties - Advanced, Correlation
IMAP	• General - Script • Network - Port Mapping
Java over HTTP	• General - Recording • Java Environment Settings - Java VM, Classpath • Network - Port Mapping • HTTP Properties - Browser, Recording Proxy, Advanced, Correlation
Java Record Replay	• Java Environment Settings - Java VM, Classpath • Recording Properties - Recorder Options, Serialization Options, Correlation Options, Log Options, Corba Options
Java Vuser	• None
LDAP	• General - Script
MMS (Media Player)	• None
Microsoft.NET	• General - Script • .NET - Recording, Filters
Microsoft Remote Desktop Protocol (RDP)	• General - Script • RDP - Client Startup, Code Generation (Basic, Advanced, and Agent) • Network - Port Mapping
MS Exchange (MAPI)	• None

MMS (Multimedia Messaging Service)	• General - Script
ODBC	• General - Script • Database - Database
Oracle (2-Tier)	• General - Script • Database - Database
Oracle NCA	• General - Script, Protocols, Recording • Network - Port Mapping • HTTP Properties - Advanced, Correlation
Oracle Web Applications 11i	• General - Script, Recording • GUI Properties - Advanced, Web Event Configuration • Network - Port Mapping • HTTP Properties - Advanced, Correlation
PeopleSoft Enterprise	• General - Script, Recording • GUI Properties - Advanced, Web Event Configuration • Network - Port Mapping • HTTP Properties - Advanced, Correlation
PeopleSoft Tuxedo	• None
POP3	• General - Script • Network - Port Mapping
Real	• None
SAP Web	• General - Script, Recording • Network - Port Mapping • HTTP Properties - Advanced, Correlation
SAP Click & Script	• General - Script, Recording • GUI Properties - Advanced, Web Event Configuration • Network - Port Mapping • HTTP Properties - Advanced, Correlation
SAP GUI	• General - Script • SAP GUI - General, Code Generation, Auto Logon

Siebel Web	• General - Script, Protocols, Recording • Network - Port Mapping • HTTP Properties - Advanced, Correlation
Silverlight	• General - Script, Protocols, Recording • Silverlight - Services • Network - Port Mapping • HTTP Properties - Advanced, Correlation • Data Format Extensions - Chain Configuration, Code Generation
SMTP	• General - Script • Network - Port Mapping
Tuxedo, Tuxedo 6	• None
VB Script Vuser	• None
VB Vuser	• None
Web (Click & Script)	• General - Script, Recording • GUI Properties - Advanced, Web Event Configuration • Network - Port Mapping • HTTP Properties - Advanced, Correlation
Web (HTTP/HTML)	• General - Script, Protocols, Recording • Network - Port Mapping • HTTP Properties - Advanced, Correlation • Data Format Extension - Chain Configuration, Code Generation
Web Services	• General - Script, Protocols, Recording • Traffic Analysis - Traffic Filters • Network - Port Mapping • HTTP Properties - Advanced, Correlation
Windows Sockets	• Sockets - Winsock

Citrix > Configuration Node

Enables you to set the window properties and encryption settings for the Citrix client during the recording session.

To access	**VuGen > Record > Recording Options > Citrix > Configuration**

Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Encryption Level	The level of encryption for the ICA connection: **Basic**, **128 bit for login only**, **40 bit**, **56 bit**, **128 bit**, or **Use Server Default** to use the machine's default.
Window Size	The size of the client window. **Default value:** 800 x 600.

Citrix > Code Generation

Enables you to configure the way VuGen captures information during recording.

To access	**VuGen > Record > Recording Options > Citrix > Code Generation**
Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • Text synchronization steps that you add manually during the recording are not affected by the above settings—they appear in the script even if you disable the above options. • The synchronization options also work for regenerating a script. For example, if you originally recorded a script with **Add text synchronization calls** disabled, you can regenerate after to recording to include text synchronization.

User interface elements are described below:

UI Element	Description
Use Citrix Agent input in Code Generation	Use the Citrix Agent input to generate a more descriptive script with additional synchronization functions. **Default value:** enabled. • **Add text synchronization calls.** Adds text synchronization **Sync on Text** steps before each mouse click. **Default value:** disabled.

Citrix > Login Node

Enables you to you set the connection and login information for the recording session.

To access	**VuGen > Record > Recording Options > Citrix > Login**

Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • If you do not provide login information, you are prompted for the information when the client locates the specified server.

User interface elements are described below:

UI Element	Description
Connection	• **Network Protocol**. The preferred protocols are TCP/IP and TCP/IP+HTTP. Most Citrix Servers support TCP/IP, however Citrix Clients starting with 11.2 do not. Certain servers, however, are configured by the administrators to allow only TCP/IP with specific HTTP headers. If you encounter a communication problem, select the TCP/IP+HTTP option. • **Server.** The Citrix server name. To add a new server to the list, click **Add**, and enter the server name (and its port for TCP/IP + HTTP). **Note:** Multiple servers apply only when you specify a Published Application. If you are connecting to the desktop without a specific application, then list only one server. • **Published Application.** The name of the **Published Application** as it is recognized on Citrix server. The drop-down menu contains a list of the available applications. If you do not specify a published application, VuGen uses the server's desktop. If you added or renamed a published application, close the Recording options and reopen them to view the new list. Additionally, you can also enter the name of a published application manually if you know it exists (useful in cases where the drop-down list is inaccurate). To change the name of the published application on the Citrix client, you must make the change on the Citrix Server machine. Select **Manage Console > Application** and create a new application or rename an existing one. **Note**: If you do not specify a published application, Citrix load balancing will not work. To use load balancing when accessing the server's desktop, register the desktop as a published application on the server machine, and select this name from the Published Application drop-down list.
Define connection parameters	Allows you to manually define the logon and connection details.
Logon Information	Specify the **User Name**, **Password**, and **Domain** of the citrix user. Optionally, you can also specify the **Client Name** by which the MetaFrame server identifies the client.

Use ICA file for connection parameters	Specify an ICA file with the log configuration information.

Citrix > Recorder Node

Enables you to specify how to generate window names where the window titles change during recording. You can also specify whether to save snapshots of the screens together with the script files and whether to generate text synchronization functions.

To access	**VuGen > Record > Recording Options > Citrix > Recorder**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Save snapshots	Saves a snapshot of the Citrix client window for each script step, when relevant. We recommend that you enable this option to provide you with a better understanding of the recorded actions. Saving snapshots, however, uses more disk space and slows down the recording session.
Window name	In some applications, the active window name changes while you are recording. If you try to replay the script as is, the Vuser uses the original window name and the replay may fail. You can specify a naming convention for the windows in which VuGen uses a common prefix or common suffix to identify the windows as follows: • **Use new window name as is.** Set the window name as it appears in the window title. (default) • **Use common prefix for new window names.** Use the common string from the beginning of the window titles as a window name. • **Use common suffix for new window names.** Use the common string from the end of the window titles as a name. Alternatively, you can modify the window names in the actual script after recording. In the Script view, locate the window name, and replace the beginning or end of the window name with the "*" wildcard notation. **Example:** ctrx_sync_on_window ("My Application*", ACTIVATE, ...CTRX_LAST);

COM/DCOM > Filter Node

Enables you to define which COM/DCOM objects to record.

To access	Use one of the following: • **VuGen > Record > Recording Options > COM/DCOM > Filter** • **VuGen > Replay > Recording Options > COM/DCOM > Filter**

User interface elements are described below:

UI Element	Description
DCOM Profile	**Specify one of the following filter types:** • **Default Filter.** The filter to be used as the default when recording a COM Vuser script. • **New Filter.** A clean filter based on the default environment settings. Note that you must specify a name for this filter before you can record with its settings. You can also save the current settings and delete a filter using the **Save As** and **Delete** buttons.
DCOM Listener Settings List	Displays a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class. To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item. An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, a dialog box opens asking you if you also want to exclude the interface in all classes that implement it this interface. Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box. • **Environment.** The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record. • **Type Libraries.** A type library .**tlb** or .**dll** file, that represents the COM object to record. All COM objects have a type library that represents them. You can select a type library from the Registry, Microsoft Transaction Server, or file system. **Type Libraries.** In the lower section of the dialog box, VuGen displays the following information for each type library. • **TypLib.** The name of the type library (tlb file). • **Path.** The path of the type library. • **Guid.** The Global Unique Identifier of the type library.

Add	Adds another COM type library. • **Browse Registry.** Displays a list of type libraries found in the registry of the local computer. Select the check box next to the desired library or libraries and click **OK**. • **Browse file system**. Allows you to select type libraries from your local file system. • **Browse MTS**. add a component from a Microsoft Transaction Server. The MTS Components dialog box prompts you to enter the name of the MTS server. Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine. Select one or more packages of MTS components from the list of available packages and click **Add**. Once the package appears in the list of Type Libraries, you can select specific components from the package.
Remove	Removes a COM type library.
Exclude...	Excludes interfaces in the filter through the Excluded Interfaces dialog box. In this dialog box, the checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click **Add Interface...** in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the **Add Interface...** feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter. An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the a warning. If you check **Don't ask me again** and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click **Yes to all** to change the status of all instances of this interface for all other classes, click **No to all** to leave the status of all other instances unchanged. Click **Next Instance** to view the next class that uses this interface.

COM/DCOM > Options Node

Enables you to set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.

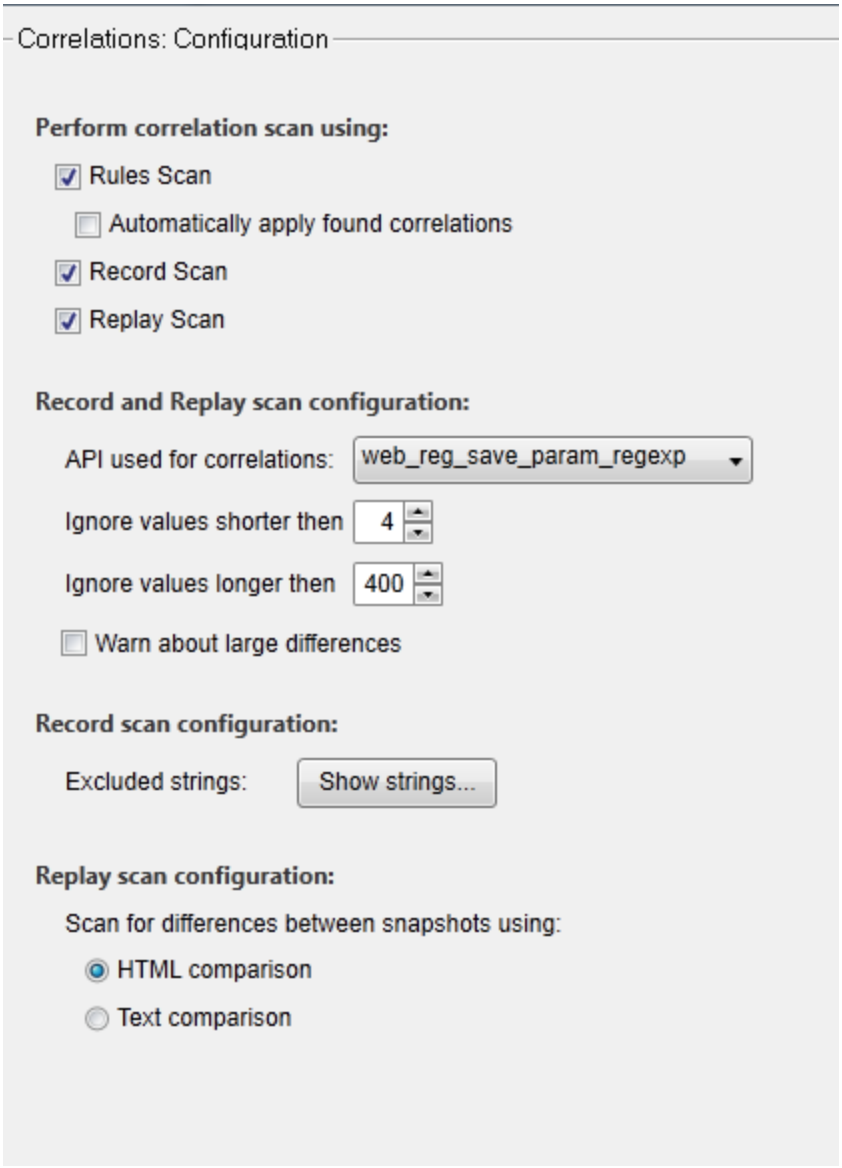
To access	**VuGen > Record > Recording Options > COM/DCOM > Options**

User interface elements are described below:

UI Element	Description
ADO Recordset filtering	Condense multiple recordset operations into a single-line fetch statement (enabled by default).
Declare Temporary VARIANTs as Globals	Define temporary VARIANT types as Globals, not as local variables (enabled by default).
Fill array in separate scopes	Fill in each array in a separate scope (enabled by default).
Fill structure in separate scopes	Fill in each structure in a separate scope (enabled by default).
Generate COM exceptions	Generate COM functions and methods that raised exceptions during recording (disabled by default).
Generate COM statistics	Generate recording time performance statistics and summary information (disabled by default).
Limit size of SafeArray log	Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).
Release COM Objects	Record the releasing of COM objects when they are no longer in use (enabled by default).
Save Recordset content	Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).
Trap binded moniker objects	Trap all of the bound moniker objects (disabled by default).

Correlations > Configuration

This pane enables you to configure settings for the Correlation tab.

UI example	
To access	**VuGen > Recording Options > Correlations > Configuration**
Important information	• "Correlation Tab [Design Studio] Overview" on page 134 • "Correlations > Rules" on page 276 • This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
Relevant tasks	"How to Correlate Scripts Using Design Studio" on page 137

User interface elements are described below:

UI Element	Description
Scan for correlations applying:	
Rules Scan	Apply correlation rules when performing correlation scan. For details, see "Correlations > Rules" on next page.
Automatically correlate values found	Design Studio will automatically correlate dynamic values found using the rules scan.
Record Scan	Scan for correlations with the record based engine.
Replay Scan	Scan for correlations with the replay based engine.
Record and Replay scan configuration	
API used for correlations	Select the API function to be used for correlation: **Boundary based**: web_reg_save_param_ex **Regular Expression**: web_reg_save_param_regexp
Exclude strings	Enables you to enter strings that should be ignored by the record and replay scan.
Ignore values shorter than [ ]	Enables you to define how short a dynamic value can be before it is ignored by the record or replay scan. Default length is 4 characters.
Ignore values longer than [ ]	Enables you to define how long a dynamic value can be before it is ignored by the record or replay scan. Default length is 400 characters.
Warm me if the dynamic string size is greater than 10 KB	Issues a warning if you try to correlate a string whose size is 10 KB or larger.
Record scan configuration	
Heuristic level	Enables you to set the filter level that controls the amount of correlation results that are returned. The higher the filter level, the longer the scan will take to run. **High. Design Studio** performs a detailed scan returning a highly refined result set. This is default setting. **Medium. Design Studio** performs a less detailed scan returning more results. **Low. Design Studio** performs a quick scan returning the most results.
Record scan configuration	

UI Element	Description
Replay scan configuration	
Scan for differences between snapshots using	Select a comparison method: • **HTML Comparison.** Display the differences in HTML code only. • **Text Comparison.** Display all text, HTML, and binary differences.

Correlations > Rules

This dialog box enables you to manage correlation rules that automatically correlate dynamic values during code generation. You can:

- Add a new application
- Add a new rule
- Delete a rule
- Export rules
- Import rules
- Test a rule

To access	Do one of the following: • **VuGen > Record > Recording Options > Correlations > Rules** • **Studio** > **Correlations > Rules**
Important information	"Correlation Overview" on page 133 "Correlation Tab [Design Studio] Overview" on page 134
Relevant tasks	"How to Correlate Scripts Using Design Studio" on page 137

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Application List >	A list of applications and their rules. • Select the check box adjacent to the application to activate it during recording. • Expand the application tree to select the checkbox adjacent to the rules to activate them during recording.

UI Element	Description
New Application	Add a new application to **<Application List>**.
New Rule	Enter a new rule for the selected application in Correlation Rules. For details, see "New Rule Pane" below.
Delete	Delete the selected application or rule from the list.
Import...	Import a file containing correlation rule definitions.
Export...	Export a file containing a correlation rule definition.
Test...	Test a correlation rule. For details, see "Token Substitution Testpad Dialog Box" on page 279.

New Rule Pane

Enables you to define a new custom rule.

To access	**VuGen > Record > Recording Options > Correlation > Rules > New Rule**
Important information	This pane is available only for specific protocols. For a complete list of protocols and their associated nodes, see the "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Advanced...	Opens the "Advanced Correlation Properties Dialog Box" on next page.

Action	Specify the type of action for the rule from the following options: • **Search for Parameters in all of the Body Text.** Searches the entire body—not just links, form actions or cookies. It searches the text for a match using the borders that you specify. • **Search for parameters inlinks and form actions.** Searches within links and forms' actions for the text to parameterize. This method is for application servers where you know the context rules. You define a left boundary, a right boundary, an alternate right boundary, and an instance of the left boundary within the current link. • **Search for Parameters from cookie headers.** Similar to the previous rule, except that the value is extracted from cookie text (exactly as it appears in the recording log) instead of from a link or form action. • **Parameterize form field value.** Saves the named form field value to a parameter. It creates a parameter and places it in the script before the form's action step. For this option, you need to specify the field name. • **Text to enter a web_reg_add_cookie function by** method inserts a **web_reg_add_cookie** function if it detects a certain string in the buffer. It only adds the function for those cookies with the specified prefix. For this option, you need to specify the search text and the cookie prefix.
Scan Type	Specify the scan type either regular expression or boundary based.
RegExp String	Specify the regular expression. This element only applies to a regular expression scan type.
Left boundary	The left-most boundary where the rule will apply. This element only applies to boundary based scan type.
Match Case	Matches the case when looking for boundaries.
Parameter prefix	Uses a prefix in all automatically generated parameters based on this rule. Prefixes prevent you from overwriting existing user parameters. In addition, prefixes allow you to recognize the parameter in your script. For example, in Siebel Web, one of the built-in rules searches for the **Siebel_row_id** prefix.
Right boundary	The right-most boundary where the rule will apply. Use the drop-down menu to define this boundary as either the end of a string, a newline character, or a user-defined text. The element only applies to boundary based scan type.
Use '#' for any digit	Replaces all digits with a hash sign. The hash signs serve as wildcard, allowing you to find text strings with any digit. **Example:** If you enable this option and specify **HP###** as the left boundary, **HP193** and **HP284** will be valid matches. This element only applies to boundary based scan type.

Advanced Correlation Properties Dialog Box

Enables you to set the advanced options for correlation rules.

To access	**VuGen > Record > Recording Options > Correlation > Rules > New Rule > Advanced**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see the "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Alternate right boundary	Alternative criteria for the right boundary, if the previously specified boundary is not found. Select one of the following options: **User-defined Text, Newline Character, End Of Page.**
Always create new parameter	Creates a new parameter for this rule even if the value replaced by the parameter has not changed from the previous instance.
Left boundary instance	The number of occurrences of the left boundary in order for it to be considered a match.
Length	The length of the string, starting with the offset, to save to the parameter. If this is not specified, the parameter continues until the end of the found value.
Offset	The offset of the string within the found value.
Replace with parameter only for exact matches	Replaces a value with a parameter only when the text exactly matches the found value.
Reverse search	Performs a backwards search.

Token Substitution Testpad Dialog Box

Enables you to test correlation rules before applying them.

To access	**VuGen > Record > Recording Options > Correlations > Rules > Test**

User interface elements are described below:

UI Element	Description
	Runs the test.
Applied rules	A list of the rules that were applied during the test.
Source string for substitution	Enter the source string for substitution.
Substitution Result	The results of the test.

Database > Database Node

Enables you to set the recording options for database protocols.

To access	**VuGen > Record > Recording Options > Database > Database**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Advanced...	Opens the "Database > Advanced Recording Options Dialog Box" below.
Automatic transactions	Marks every **lrd_exec** and **lrd_fetch** function as a transaction. When these options are enabled, VuGen inserts **lr_start_transaction** and **lr_end_ transaction** functions around every **lrd_exec** or **lrd_fetch function. Default value:** Disabled.
Script options	Generates comments into recorded scripts, describing the **lrd_stmt** option values. In addition, you can specify the maximum length of a line in the script. **Default value:** 80 characters**.**
Think time	VuGen automatically records the operator's think time. You can set a threshold level, below which the recorded think time will be ignored. If the recorded think time exceeds the threshold level, VuGen places an **lr_think_time** statement before LRD functions. If the recorded think time is below the threshold level, an **lr_think_time** statement is not generated. **Default value:** five seconds.

Database > Advanced Recording Options Dialog Box

Enables you set the advanced recording options for database protocols.

To access	**VuGen > Record > Recording Options > Database > Database > Advanced**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Code generation buffer size	Specify in kilobytes the maximum size of the code generation buffer. **Default value:** 128 kilobytes.

CtLib Function	You can instruct VuGen to generate a send data time stamp or an extended result set statement. • **Generate send data time stamp.** Generates **lrd_send_data** statements with the **TotalLen** and **Log** keywords for the **mpszReqSpec** parameter. The Advanced Recording Options dialog box lets you instruct VuGen to also generate the **TimeStamp** keyword. If you change this setting on an existing script, you must regenerate the Vuser script by choosing **Record > Regenerate Script.** It is not recommended to generate the **Timestamp** keyword by default. The timestamp generated during recording is different than that generated during replay and script execution will fail. You should use this option only after a failed attempt in running a script, where an **lrd_result_set** following an **lrd_send_data** fails. The generated timestamp can be correlated with a timestamp generated by an earlier **lrd_send_data**. • **Generate extended result set statement.** Generates an **lrd_result_set** function when preparing the result set. This setting instructs VuGen to generate the extended form of the **lrd_result_set** function, **lrd_result_set_ext**. In addition to preparing a result set, this function also issues a return code and type from **ct_results**.
Recording engine	You can instruct VuGen to record scripts with the older LRD recording engine for compatibility with previous versions of VuGen. **Note:** This option is available only for single-protocol scripts.
Recording log options	You can set the detail level for the trace and ASCII log files. The available levels for the trace file are **Off**, **Error Trace**, **Brief Trace**, or **Full Trace**. The error trace only logs error messages. The Brief Trace logs errors and lists the functions generated during recording. The Full Trace logs all messages, notifications, and warnings. You can also instruct VuGen to generate ASCII type logs of the recording session. The available levels are **Off**, **Brief detail**, and **Full detail**. The Brief detail logs all of the functions, and the Full detail logs all of the generated functions and messages in ASCII code.

Data Format Extension > Chain Configuration Node

Enables you to add, remove, and modify chains and Data Format Extensions. You can add a Data Format Extension to an extension chain.

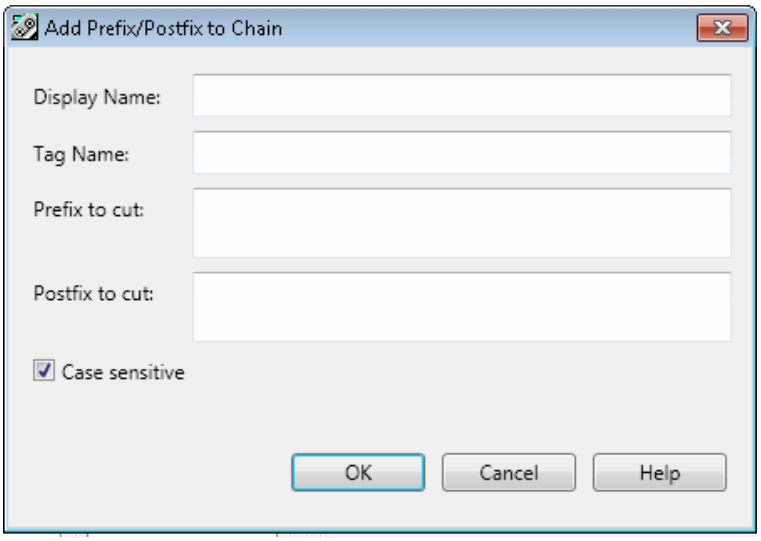
To access	Do one of the following: • **VuGen > Record > Recording Options > Data Format Extension > Chain Configuration** • **VuGen > Replay > Run Time Settings > Data Format Extensions > Code Generation**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
See also	"Data Format Extensions" on page 675 "Data Format Extension List" on page 683

User interface elements are described below:

UI Element	Description
Chains pane	Displays a list of the chains.
	**Add Chain**. Enables you to add a new chain.
	**Edit Chain Name**. Enables you to modify the name of the chain.
	**Delete Chain**. Removes the selected chain.
Chain: <chain name> pane	
	**Add DFE**. Enables you to add a Data Format Extension to the selected chain in the **Chains** pane. For more information on Data Format Extensions, see "Data Format Extension List" on page 683.
	**Edit DFE**. If you selected a Prefix/Postfix Extension, you can edit the details in the **Add Prefix/Postfix to Chain** dialog box. For more information on the dialog box, see " Add Prefix/Postfix to Chain Dialog Box" below.
	**Delete DFE**. Removes the selected Data Format Extension from the chain.
	**Move Up/Down**. Moves a Data Format Extension up or down in the chain. Extensions are run in the order in which they appear in the extensions list.
Name	The display name of the Data Format Extension.
Tag	The unique ID of the extension.
Provider	The creator of the Data Format Extension.
Continue Processing	Determines how the chain behaves if the Data Format Extension successfully reformats the data. • If the condition is true, VuGen continues to pass the converted data to the next Data Format Extension in the chain. • If the condition is false, the chain is terminated.

Add Prefix/Postfix to Chain Dialog Box

This dialog enables you to add or edit a prefix/postfix extension to the selected chain.



To access	1. Go to **Recording > Recording Options > Data Format Extension > Chain Configuration** node. 2. In the **Chain: <Chain name>** area, click the button. 3. Select the **Prefix/Postfix Extension** option.
See also	"Data Format Extension > Chain Configuration Node" on page 281 "Data Format Extensions" on page 675 "Data Format Extension List" on page 683

User interface elements are described below:

UI Element	Description
Case sensitive	Sets the extension to cut from the defined prefix and postfix of the string only if the letter cases match.
Display name	The name of Prefix/Postfix Extension.
Postfix to cut	The section you want to cut, from the end of the string.
Prefix to cut	The section you want to cut, from the beginning of the string.
Tag name	The unique ID of the Prefix/Postfix Extension.

Add Data Format Extension

This dialog box enables you to select the data format extension type.

UI example	
To access	**VuGen > Recording Options > Chain Configuration >** 
Important information	• "Data Format Extension > Chain Configuration Node" on page 281 • " Add Prefix/Postfix to Chain Dialog Box" on page 282
Relevant tasks	"How to Apply Chains to Sections of the HTTP Message" on page 681

Data Format Extension	Description
Base64 Extension	Decodes strings that are encoded with as BASE64 encoder.
GWT Extension	Transforms GWT data to XML format.
URL Encoding Extension	Decodes strings that are encoded with URL encoding format.
JSON Extension	Transforms JSON data to XML format.
XML Validator Extension	Receives data and checks to see if it conforms with XML syntax. This check allows VuGen to perform correlations based on XPath and to display snapshot data in an XML viewer.
Prefix Postfix Extension	Enables you to cut data from the beginning and/or end of a string which you do not want decoded. You can add and customize as many prefix/postfix extensions as required. Each postfix/prefix extension created should have a unique display name and tag name.
Binary XML Extension	Transforms Microsoft WCF binary XML into XML format.
Remedy Extension	Transforms Remedy request data into XML format. Note that this extension does not transform Remedy response data - which is JavaScript code.
XSS Extension	Enables you to test sites that use Cross Site Scripting (XSS) defense code.

Data Format Extension > Code Generation Node

Enables Data Format Extensions during code generation. Enables you to define chains for each message section of the HTTP message.

To access	**VuGen > Record > Recording Options > Data Format Extensions > Code Generation**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
See also	"Data Format Extensions" on page 675 "Data Format Extension List" on page 683

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Enable data format extension	Enables you to select chains for each message section of the HTTP message. Deselected by default.
Configuration	
Format	• **Code and snapshots**. (default) Enables Data Format Extensions on the code and snapshot data. • **Snapshots**. Enables Data Format Extensions on snapshot data, but does not format the data in the script itself.
Verify formatted data	Checks the results of the formatted data by converting it back to the original state and verifying that it matches the original data. **Note**: Available for Base64 extension only.
Chains	
	Imports the Data Format Extensions from a file.
	Exports the Data Format Extensions to a file.
<Message sections list>	Displays a list of the following sections of the HTTP message included in the script: • Body • Headers • Cookies • Query String When you select a message section from the list, the title of the section chains pane (described below) reflects your selection and the pane displays the list of chains for that section.
<Section Chains>	

	**Add Chain**. Adds chain to selected message section. **Note**: • Enabled for Headers and Cookies sections only. Enables you to add additional chains to the selected message section. • For VuGen to correctly match the chain to the Headers or Cookies section, the name in the **Name** column must match the name of the Headers or Cookies section.
	**Delete Chain**. Removes chain from corresponding message section. **Note:** You cannot delete the default options from any of the message sections.
	**Reset**. Clears the selected chain in the Chain column.

EJB > Code Generation Options Node

Enables you to set the EJB recording options.

To access	**VuGen > Record > Recording Options > EJB Options > Code Generation Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Auto Transaction	Automatically marks all EJB methods as transactions. This encloses all methods with **lr.start_transaction** and **lr.end_transaction** functions. **Default value:** enabled.
EJB Initialization Method	The method to which the EJB/JNDI initialization properties are written. The available methods are **init** and **action**. **Default value:** init.
Insert Value Check	Automatically insert an **lr.value_check** function after each EJB method. This function checks for the expected return value for primitive values and strings.

Flex > RTMP Node

This node enables you to include the flex_rtmp_recive_stream step in Flex RTMP scripts.

To access	**VuGen > Record > Recording Options > Flex > RTMP**
Important Information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Generate flex_ rtmp_ receive_ stream step	Generates a single step when recording a stream. This step does not replay certain actions, such as pause and seek. If your script requires these actions, clear the check box to record all receive and send steps. However, in this case, you must manually modify your script as described in the *Readme*.

Flex > Configuration Node (Recording)

Enables you to set an external JVM (Java Virtual Machine) path.

To access	**VuGen > Record > Recording Options> Flex > Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Use External JVM	Enables you to use external JVM. VuGen must be restarted for the changes to be applied.
External JVM Path	The path of the external JVM.
UseGraniteDS configuration	Defines the server side Data Service configuration. If you select this option, do not select **Use Flex LCDS/BlazeDS jars** to serialize the messages.

Flex > Code Generation Node

Enables you to set the code generation options for the Flex protocol.

To Access	**Record > Recording Options > Flex > Code Generation**
Important Information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Encode AMF3 using external parser	Attempt to encode externalizable objects using an external parser.

Encode externalizable object with LoadRunner parser	When recording Flex applications, in certain cases VuGen may be unable to decode externalizable objects. This is due to a proprietary encoding scheme employed by the Flex 2 application. To overcome this issue, VuGen generates Custom Request functions containing unparsed AMF3 binary data when it encounters an externalizable object. You can attempt to encode these objects using the LoadRunner parser as well. This decodes all externalizable objects as standard AMF3 objects, generating **flex_amf_call** functions. However, if objects cannot be parsed, they are not decoded at all so you have to check the script after it is generated to make sure that the parser was effective. Additionally, this option may reduce the stability of code generation. **Default value:** enabled.
Flex server/application JAR files location	The location of the external parser and JAR files to be encoded.

Flex > Externalizable Objects Node (Recording)

This dialog box enables you to configure how LoadRunner handles externalizable objects in Flex scripts.

To access	**VuGen > Record > Recording Options > Flex > Externalizable Objects**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
Relevant tasks	• "How to Serialize Using External Java Serializer" on page 524 • "How to Serialize Scripts with the LoadRunner Serializer" on page 525
See also	• "Flex Overview" on page 512 • " Externalizable Objects in Flex Scripts" on page 521

User interface elements are described below:

UI Element	Description
Do not serialize externalizable objects	Generate script using default settings.

Serialize objects using	Select the appropriate option: • Select **LoadRunner AMF serializer** if you are not using the Adobe LiveCycle Data Services or Adobe BlazeDS server. • Select **Custom Java classes** and select one or both of the available options: ▪ Select **Use Flex LCDS/BlazeDS jars** if you are using Flex LCDS or BlazeDS jars to serialize the messages. If you selected **UseGraniteDS configuration** in the **Configuration** node, do not select **Use Flex LCDS/BlazeDS jars**. ▪ Select **Use additional jars** to add additional jars to serialize the messages. You must copy the jar files from the server and specify their location in the **Classpath Entries** list described below. Copy only those jars that contain the class that is externalizable. Ensure that the files exist in the same location on all load generator computers. If you add jars with the same names as the Flex LCDS or Blaze DS jars chosen by selecting the first check-box, these files will be overwritten.
Classpath Entries List	
	**Down Arrow.** Moves a classpath entry down the list.
	**Up Arrow.** Moves a classpath entry up the list.
	**Add Classpath.** Adds a new line to the classpath list.
	**Add Classpath Folder.** Adds all files from the folder to the classpath list.
	**Delete.** Permanently removes a classpath.

General > Protocol Node

Enables you to set the script generation preferences by setting the scripting language and options.

To access	**VuGen > Record > Recording Options > General > Protocols**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Active Protocols List	A list of the protocols which comprise your multiple protocol script. VuGen lets you modify the protocol list for which to generate code during the recording session. Select the check boxes adjacent to the protocols you want to record in the next recording session. Clear the check boxes adjacent to the protocols you do not want to record in the next recording session.

General > Code Generation

This pane of the Recording Options dialog box enables you to define what tasks VuGen performs automatically after generating a Vuser script.

To access	**VuGen > Record > Recording Options > General > Code Generation**
Relevant tasks	"How to Create an Asynchronous Vuser Script" on page 176 "How to Correlate Scripts Using Design Studio" on page 137

User interface elements are described below:

UI Element	Description
Correlations Scan	Instructs VuGen to analyze the Vuser script to locate dynamic values that may need to be correlated. This scan is performed after a new script is recorded and after an existing script is regenerated.
Async Scan	Instructs VuGen to analyze the Vuser script to locate asynchronous communication. This scan is performed after a new script is generated and after an existing script is regenerated.
Async Options...	Opens the "Asynchronous Request Thresholds Dialog Box" on page 194.

General > Recording Node

Enables you to specify what information to record and which functions to use when generating a Vuser script, by selecting a recording level.

To access	**VuGen > Record > Recording Options > General > Recording**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
HTML Advanced...	Opens the "Advanced HTML Dialog Box" on next page.
URL Advanced...	Opens the "Advanced URL Dialog Box" on next page.
GUI-based script	Recommended for most applications, including those with JavaScript. Also recommended for PeopleSoft Enterprise and Oracle Web Applications.

HTML-based script	This is the default recording level for Web (HTTP/HTML) Vusers. It instructs VuGen to record HTML actions in the context of the current Web page. It does not record all resources during the recording session, but downloads them during replay. This options is recommended for browser applications with applets and VB script.
URL-based script	Record all requests and resources from the server. It automatically records every HTTP resource as URL steps (**web_url** statements), or in the case of forms, as **web_submit_data**. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions, nor does it record frames. This options is recommended For non-browser applications.

Advanced URL Dialog Box

Enables you to set the advanced options for scripts using the URL recording mode.

To access	**VuGen > Record > Recording Options > General > Recording > URL Advanced**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Use Defaults	Restores the default settings of this dialog box.
Create concurrent groups for resources after their source HTML page	Records the resources in a concurrent group (enclosed by **web_concurrent_start** and **web_concurrent_end** statements) after the URL. Resources include files such as images and **js** files. If you disable this option, the resources are listed as separate **web_url** steps, but not marked as a concurrent group.
Enable EUC-Encoded Web Pages	(For Japanese windows only) Instructs VuGen to use EUC encoding. For more information, see "EUC-Encoding (Japanese Windows only)" on page 259.
Use web_custom_request only	Records all HTTP requests as custom requests. VuGen generates a **web_custom_request** function for all requests, regardless of their content. Recommended for non-browser applications.

Advanced HTML Dialog Box

Enables you to set the advanced options for HTTP-based scripts.

To access	**VuGen > Record > Recording Options > General > Recording > HTML Advanced**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Use Defaults	Restores the default settings of this dialog box.
Non-HTML generated elements	Many Web pages contain non-HTML elements, such as applets, XML, ActiveX elements, or JavaScript. These non-HTML elements usually contain or retrieve their own resources. Using the following options, you can control how VuGen records non HTML-generated elements. • **Record within the current script step.** Does not generate a new function for each of the non HTML-generated resources. It lists all resources as arguments of the relevant functions, such as **web_url**, **web_link**, and **web_submit_data**. The resources, arguments of the Web functions, are indicated by the **EXTRARES** flag. • **Record in separate steps and use concurrent groups**. Creates a new function for each one of the non HTML-generated resources and does not include them as items in the page's functions (such as **web_url** and **web_link**). All of the **web_url** functions generated for a resource are placed in a concurrent group (surrounded by **web_concurrent_start** and **web_concurrent_end**). • **Do not record.** Does not record any non-HTML generated resources.
Script type	• **A script describing user actions.** Generates functions that correspond directly to the action taken. It creates URL (**web_url**), link (**web_link**), image (**web_image**), and form submission (**web_submit_form**) functions. The resulting script is very intuitive and resembles a context sensitive recording. • **A script containing explicit URL's only.** Records all links, images and URLs as **web_url** statements, or in the case of forms, as web_submit_data. It does not generate the **web_link**, **web_image**, and **web_submit_form** functions. The resulting script is less intuitive. This mode is useful for instances where many links within your site have the same link text. If you record the site using the first option, it records an ordinal (instance) for the link, but if you record using the second option, each link is listed by its URL. This facilitates parameterization and correlation for that step.

General > Script Node

Enables you to set the script generation preferences by setting the scripting language and options.

To access	**VuGen > Record > Recording Options > General > Script**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Close all AUT processes when recording stops	Automatically closes all of the AUT's (Application Under Test) processes when VuGen stops recording. **Default value:** disabled.
Correlate arrays	Tracks and correlates arrays of all data types, such as string, structures, numbers, and so on. **Default value:** enabled.
Correlate large numbers	Correlates long data types such as integers, long integers, 64-bit characters, float, and double. **Default value:** disabled.
Correlate simple strings	Correlates simple, non-array strings and phrases. **Default value:** disabled.
Correlate small numbers	Correlates short data types such as bytes, characters, and short integers. **Default value:** disabled.
Correlate structures	Tracks and correlates complex structures. **Default value:** enabled.
Declare primitives as locals	Declares primitive value variables as local variables rather than class variables (C, C#, and .NET only). **Default value:** enabled.
Explicit variant declaration	Declares variant types explicitly in order to handle ByRef variants (Visual Basic for Applications only). **Default value:** enabled.
Generate fixed think time after end transaction	Adds a fixed think time, in seconds, after the end of each transaction. When you enable this option, you can specify a value for the think time. **Default value:** disabled, 3 seconds when enabled.
Generate recorded events log	Generates a log of all events that took place during recording. **Default value:** disabled.

, continued

Generate think time greater than threshold	Uses a threshold value for think time. If the recorded think time is less than the threshold, VuGen does not generate a think time statement. You also specify the threshold value. The default values is 3—if the think time is less than 3 seconds, VuGen does not generate think time statements. If you disable this option, VuGen will not generate any think times. **Default value:** enabled, 3 seconds.
Insert output parameters values	Inserts output parameter values after each call (C, C#, and .NET only). **Default value:** disabled.
Insert post-invocation info	Insert informative logging messages after each message invocation (non-C only). **Default value:** enabled.
Insert pre-invocation info	Insert informative logging messages before each message invocation (non-C only). **Default value:** enabled.
Maximum number of lines in action file	Create a new file if the number of lines in the action exceeds the specified threshold. The default threshold is 60000 lines (C, C#, and .NET only). **Default value:** disabled.
Replace long strings with parameter	Save strings exceeding the maximum length to a parameter. This option has an initial maximum length of 100 characters. The parameters and the complete strings are stored in the **lr_strings.h** file in the script's folder in the following format: **const char <paramName_uniqueID> ="string".** This option allows you to have a more readable script. It does not effect the performance of the script. **Default value:** enabled.
Reuse variables for primitive return values	Reuse the same variables for primitives received from method calls. This overrides the **Declare primitives as locals setting**. **Default value:** enabled.
Track processes created as COM local servers	Track the activity of the recorded application if one of its sub-processes was created as a COM local server (C and COM only). **Default value:** enabled.
Use full type names	Use the full type name when declaring a new variable (C# and .NET only). **Default value:** disabled.

, continued

Use helpers for arrays	Use helper functions to extract components in variant arrays (Java and VB Scripting only). **Default value:** disabled.
Use helpers for objects	Use helper functions to extract object references from variants when passed as function arguments (Java and VB Scripting only). **Default value:** disabled.
Use protected application recording	Use this option if VuGen is unable to record your application. Your application may block access to VuGen, and recording with this option selected may enable access. **Default value:** disabled.

GUI Properties > Web Event Configuration Node

Enables you to set the level of detail recorded in a script (web event recording).

To access	**VuGen > Record > Recording Options > GUI Properties > Web Event Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Basic Event Configuration Level	• Always records click events on standard Web objects such as images, buttons, and radio buttons. • Always records the submit event within forms. • Records click events on other objects with a handler or behavior connected. • Records the mouseover event on images and image maps only if the event following the mouseover is performed on the same object.
Custom Settings	Opens the "Custom Web Event Recording Configuration Dialog Box" below, where you can customize the event recording configuration.
High Event Configuration Level	In addition to the objects recorded in the **Medium** level, it records mouseover, mousedown, and double-click events on objects with handlers or behaviors attached.
Medium Event Configuration Level	In addition to the objects recorded in the **Basic** level, it records click events on the <DIV>, <SPAN>, and <TD> HTML tag objects.

Custom Web Event Recording Configuration Dialog Box

Enables you to customize the level of web event recording.

To access	**VuGen > Record > Recording Options > GUI Properties > Web Event Configuration > Custom Settings**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Object List>	A list of the web objects. Each web object can be customized according to the other settings in this dialog box.
<Object Menu>	• **Add.** Adds a new HTML tag object to the object list. Type in the name of the tag. • **Delete.** Deletes an object from the object list.
Event Menu	• **Add.** Adds an event to the Event Name column of this object. • **Delete.** Deletes an event from the Event Name column of this object.
Event Name	A list of events associated with the object.
File Menu	• **Load Configuration.** Loads a previously created custom configuration. • **Save Configuration As.** Saves the current configuration.
Listen	The criteria which determines when VuGen listens for an event. • **Always**. Always listen to the event. • **If Handler**. Listens to the event if a handler is attached to it. A handler is code in a Web page, typically a function or routine written in a scripting language, that receives control when the corresponding event occurs. • **If Behavior**. Listens to the event if a DHTML behavior is attached to it. A DHTML behavior encapsulates specific functionality or behavior on a page. When applied to a standard HTML element on a page, a behavior enhances that element's default behavior. • **If Handler or Behavior**. Listens to the event if either a handler or a behavior is attached to it. • **Never**. Never listens to the event. For more information, see "Tips for Working with Event Listening and Recording" on page 263.

Record	The criteria which determines when VuGen records an event. • **Enabled.** Records the event each time it occurs on the object as long as VuGen listens to the event on the selected object, or on another object to which the event bubbles. Bubbling is the process whereby, when an event occurs on a child object, the event can travel up the chain of hierarchy within the HTML code until it encounters an event handler to process the event. • **Disabled.** Does not record the specified event and ignores event bubbling where applicable. • **Enabled on next event.** Same as **Enabled**, except that it records the event only if the subsequent event occurs on the same object. For example, suppose a mouseover behavior modifies an image link. You may not want to record the mouseover event each time you happen to move the mouse over this image. Because only the image that is displayed after the mouseover event enables the link event, however, it is essential that the mouseover event is recorded before a click event on the same object. For more information, see "Tips for Working with Event Listening and Recording" on page 263.
Reset Settings	Resets the custom settings to the settings of your choice: **basic**, **medium**, or **high**.

HTTP > Advanced Node

Enables you to customize the code generation settings in the area of think time, resetting contexts, saving snapshots, and the generation of **web_reg_find** functions.

To access	**VuGen > Record > Recording Options > HTTP Properties > Advanced**
Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • Some options within this node are not available in when using a multi-protocol script.

User interface elements are described below:

UI Element	Description
Headers...	Opens the "Headers Dialog Box" on page 299.
Content Types...	Opens the "Content Type Filters Dialog Box" on page 300.
Non-Resources...	Opens the "Non-Resources Dialog Box" on page 301.

<table>
<tr><td>Add comment to script for HTTP errors while recording</td><td>Adds a comment to the script for each HTTP request error. An error request is defined as one that generated a server response value of 400 or greater during recording.</td></tr>
<tr><td>Generate web_reg_find functions for page titles</td><td>Generates web_reg_find functions for all HTML page titles. VuGen adds the string from the page's title tag and uses it as an argument for web_reg_find.
Generate web_reg_find functions for sub-frames. Generates web_reg_find functions for page titles in all sub-frames of the recorded page.
Note: This option is available only for Web and Oracle NCA protocols</td></tr>
<tr><td>Parameterize server names</td><td>VuGen identifies server names and IP addresses when you regenerate a Vuser script. These server names and IP addresses are contained in specific arguments associated with specific functions in the Vuser script. [See the table below for details.] VuGen replaces the identified server names and IP addresses with parameters. Parameterizing server names and IP addresses enables you to run the Vuser script in different environments by simply changing the server and IP address values in the parameter file. For an introduction to parameters, see "Parameter Overview" on page 228.

Note: To identify data for parameterization, VuGen searches the arguments that are listed for the following functions:

<table><tr><th>API Function</th><th>Arguments</th></tr><tr><td>web_url</td><td>• URL
• Referrer</td></tr><tr><td>web_custom_request</td><td>• URL
• Referrer</td></tr><tr><td>web_image</td><td>• URL
• Referrer</td></tr><tr><td>web_submit_data</td><td>• Action
• URL
• Referrer</td></tr><tr><td>web_submit_form</td><td>• Action
• URL
• Referrer</td></tr></table>
By default, this option is not selected.</td></tr>
</table>

Record script using earlier recording engine	Record using the single-protocol recording engine. By default, for Web (HTTP/HTML) Vusers, VuGen uses the multi-protocol recording engine for all recordings even if you are only recording a single protocol.
Record think time	Records the think times and generate **lr_think_time** functions. You can also set a **Think-time Threshold** value to only generate **lr_think_time** functions when the actual think-time is greater than the threshold. **Note:** This option is available only for Wireless Protocols scripts.
Reset context for each action	Resets all HTTP contexts between actions. Resetting contexts allows the Vuser to more accurately emulate a new user beginning a browsing session. This option resets the HTML context, so that a context-less function is always recorded in the beginning of the action. It also clears the cache and resets the user names and passwords. **Note:** This option is available only for Web and Oracle NCA protocols
Save snapshot resources locally	Saves a local copy of the snapshot resources during record and replay, thereby creating snapshots more accurately and displaying them quicker.
Support charset	• **UTF-8.** Enables support for UTF-8 encoding. This instructs VuGen to convert non-ASCII UTF-8 characters to the encoding of your locale's machine in order to display them properly in VuGen. You should enable this option only on non-English UTF-8 encoded pages. The recorded site's language must match the operating system language. You cannot record non-English Web pages with different encodings (for example, UTF-8 together with ISO-8859-1 or shift_jis) within the same script. • **EUC-JP.** If you are using Japanese Windows, select this option to enable support for Web sites that use EUC-JP character encoding. This instructs VuGen to convert EUC-JP strings to the encoding of your locale's machine in order to display them properly in VuGen. VuGen converts all EUC-JP (Japanese UNIX) strings to the SJIS (Japanese Windows) encoding of your locale's machine, and adds a **web_sjis_to_euc_param** function to the script. (Kanji only)

Headers Dialog Box

Enables you to automatically send additional HTTP headers with every HTTP request submitted to the server.

To access	**VuGen > Record > Recording Options > HTTP > Advanced > Headers**
Important information	• This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • The following standard headers are considered risky: Authorization, Connection, Content-Length, Cookie, Host, If-Modified-Since, Proxy-Authenticate, Proxy-Authorization, Proxy-Connection, Referer, and WWW-Authenticate. They are not recorded unless selected in the Header list.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Plus.** Adds a new entry.
	**Minus.** Deletes an entry.
	Restores the current list to the default values and entries.
	Restores all lists to the default values and entries.
<Drop-down menu>	Controls the options for this dialog box: • **Do not record headers.** • **Record headers in list.** • **Record headers not in list.**
<Header list>	List of headers which may or may not be recorded. The lists vary depending on which drop-down item is selected. Each item can be selected or deselected using its individual checkbox.

Content Type Filters Dialog Box

Enables you to filter content types for your recorded script. You can specify the type of the content you want to record or exclude from your script.

To access	**VuGen > Record > Recording Options > HTTP > Advanced > Content Types**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Plus.** Adds a new entry.
	**Minus.** Deletes an entry.
	Restores the current list to the default values and entries.
	Restores all lists to the default values and entries.

, continued

<Drop-down menu>	Controls the options for this dialog box: • **Do not filter content types.** • **Filter content types in list.** • **Filter content types not in list.**
<Header list>	List of content types which may or may not be filtered. The lists vary depending on which drop-down item is selected. Each item can be selected or deselected using its individual checkbox.

Non-Resources Dialog Box

When you record a script, VuGen indicates whether or not it will retrieve the resource during replay using the Resource attribute in the **web_url** function. If the Resource attribute is set to 0, the resource is retrieved during script execution. If the Resource attribute is set to 1, the Vuser skips the resource type.

You can exclude specific content types from being handled as resources. For example, you can indicate to VuGen that **gif** type resources should not be handled as a resource and therefore be downloaded unconditionally.

To access	**VuGen > Record > Recording Options > HTTP > Advanced > Non-Resources**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Add.** Adds a new entry to the list.
	**Remove.** Deletes an entry from the list.
	Restores the default list.
<Non-Resource Content Type list>	List of items which should not be recorded as resources. Each item can be selected or deselected using its individual checkbox.

Java > Classpath Node

Enables you to specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper recording.

To access	**VuGen > Record > Recording Options > Java Environment Settings > Classpath**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
	**Down Arrow.** Moves a classpath entry down the list.
	**Up Arrow.** Moves a classpath entry up the list.
	**Add Classpath.** Adds a new line to the classpath list.
	**Delete.** Permanently removes a classpath.
Classpath Entries List	A list of classpath entries.

Java > VM Node

Enables you to indicate additional parameters to use when recording Java applications.

To access	**VuGen > Record > Recording Options > Java Environment Settings > Java VM**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Additional VM Parameters	List the Java command line parameters here. These parameters may be any Java VM argument. The common arguments are the debug flag (**-verbose**) or memory settings (**-ms**, **-mx**). In additional, you may also pass properties to Java applications in the form of a **-D** flag. For more information about the Java VM flags, see the JVM documentation.
Prepend CLASSPATH to -Xbootclasspath parameter	Instructs VuGen to add the Classpath before the **Xbootclasspath** (prepend the string).
Use classic Java VM	Instructs VuGen to use the classic version of VM (for example, not Sun's Java HotSpot).

Use the specified Additional VM Parameters during replay	Instructs VuGen to use the same Additional VM parameters in replay.

.NET > Filters Node

Enables you to set the recording options for .NET Vuser scripts.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Filters**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
See also	• ".NET Filters Overview" on page 591 • ".NET Filters - Advanced" on page 592 • "Guidelines for Setting .NET Filters" on page 593

User interface elements are described below:

UI Element	Description
Create...	Opens the Create a New Filter dialog box, enabling you to create a new filter. For more details, see "Create a New Filter Dialog Box [.NET Protocol]" below.
Filter Manager...	Opens the Filter Manager, allowing you to view and modify all Microsoft .NET protocol filters. For details, see "Filter Manager [.NET Protocol]" on next page.
Custom Filter	Shows the filters that you created earlier on the current machine.
Environment Filter	Lists the available environment filters: .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation of Framework 3.0).
New Filter	Indicates that you want to create a new filter.

Create a New Filter Dialog Box [.NET Protocol]

This dialog box enables you to create a new filter for .NET Vuser scripts.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Filters > New Filter > Create**

User interface elements are described below:

UI Element	Description
Based on a custom filter	Create a new filter based on a custom filter. Use the drop-down menu to select the custom filter.
Based on an environment filter	Create a new filter based on an environment filter. Use the check boxes next to the environment filters to indicate which environment filters to base the filter on.
Start with an empty filter	Create a new filter that is not based on a pre-existing filter.

Filter Manager [.NET Protocol]

This dialog box enables you to create and edit .NET filters.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Filters > Environment / Custom Filter > Filter Manager**
See also	".NET Filters Overview" on page 591 ".NET Filters - Advanced" on page 592 "Guidelines for Setting .NET Filters" on page 593

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Includes the selected element. If you manually include a parent node, the Filter Manager includes the child elements below it, provided that no other rule exists. For example, if you include a class, it will include all its methods unless you specifically excluded a method.
	Excludes the selected element. The child elements are also excluded unless they were included by another rule. By default, when you exclude a class, the Filter Manager applies the Exclude attribute to the class, but it allows the recording engine to record activity within the methods of the excluded class. When you exclude a method, however, the Filter Manager applies Totally Exclude, preventing the recording engine from recording any activity within the methods of the excluded class. Advanced users can modify these setting in the filter file.

	Removes the manual inclusion or exclusion rule. In this case, the element may be impacted by other parent elements. The inclusion and exclusion rules have the following properties: • The rules are hierarchical—if you add an include or exclude rule to a class, then the derived classes will follow the same rule unless otherwise specified. • A rule on a class only affects its public methods, derived classes, and inner classes. • A rule on a namespace affects all the classes and their public methods. • Note that adding or removing assemblies does not necessarily affect the classes that they contain—you can remove an assembly, yet its methods may be recorded due to the hierarchical nature of the filter. • As part of the filter design, several methods, such as **.cctor()** and **Dispose (bool)**, do not follow the standard hierarchal rules. **Note:** The resetting of a parent node does not override a manual inclusion or exclusion applied to a child node. For example, if you manually **exclude** a method, and then reset its class, which by default **included** all sub-nodes, your method will remain excluded. Properties and events are view-only and cannot be included or excluded through the Filter Manager. In addition, several system related elements are protected and may not be altered. For tips about including and excluding elements in the filter, see "Guidelines for Setting .NET Filters" on page 593.
	Navigates to the previous or next tree node visited by the user.
Impact Log	The Impact Log indicates what your last changes were and how they affected your filter. The user actions are listed in descending order, with the latest changes at the top. For each element affected by your manual inclusion or exclusion, the log indicates how it affected the element. It also provides a link to that element in the Filter Manager. To view the Impact Log, click the Impact Log button on the Filter Manager's toolbar or select **Actions > View** Impact Log in the Filter Manager window.
<Filter Manager Tree>	The Filter Manager tree uses symbols to illustrate the elements and their status. For details about each of the icons, see the table below. • Element icons represent the type of element—assembly, namespace, class, method, structure, property, events, or interfaces. • A check mark or X adjacent to the element icon, indicates whether or not the element is included or excluded. • A bold element indicates that it was explicitly included or excluded. This may be a result of being manually included or excluded by the user or by a pre-defined rule in the environment filter. If you reset a bold node, it returns to its original, non-bold state.

Add Reference	Opens the Add Reference dialog box with a list of .NET Framework components or assemblies in the Public Assemblies folder. For more information, see "Add Reference Dialog Box [.NET Protocol]" below.
Delete	Deletes the selected custom filter. The Filter Manager prompts you for a confirmation.
New	Opens the Create a New Filter dialog box, in which you create an empty filter or a new filter based on an existing one. For more information, see "Create a New Filter Dialog Box [.NET Protocol]" on page 303.
Remove Reference	Removes the assembly that is selected in the Filter Manager and all of the elements associated with it. The Filter Manager prompts you for a confirmation.
Save	Saves the changes you made to filter.
View Impact Log	Opens the Impact log for the selected filter. The Impact log shows which nodes in the tree were affected by recent actions.

The following table shows the Filter Manager Tree icons that represent the various elements:

Icon	Description	Icon	Description
	assembly		interface
	assembly that couldn't be loaded		method
	assembly that was partially loaded		static method
	class	{}	namespace
	constructor		property
	static constructor		static property
	event		structure
	static event		

Add Reference Dialog Box [.NET Protocol]

This dialog box enables you to add references to .NET filters.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Filters > Environment / Custom Filter > Filter Manager > Add Reference**

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description

<Component List>	A list of .NET Framework components or assemblies in the Public Assemblies folder. • To add one of the listed items, select it and click **Select**. You can select multiple components using ctrl-click. The bottom pane shows the selected references. • To add an assembly that is not in the list, click **Browse** and locate the reference on your file system or network.
<Selected Component List>	The list of selected components. The Type column indicates .**NET** for a component from the Public Assemblies folder and **File** for a component that was added by selecting **Browse**. • To clear an item from the list, select it in the bottom pane and click **Remove**.

GUI Properties > Advanced Node

Enables you to set advanced recording options for Click & Script Vusers.

To access	**VuGen > Record > Recording Options > GUI Properties > Advanced**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

Recording Settings Properties

UI Element	Description
Record rendering-related property values	Records the values of the rendering-related properties of DOM objects (for example, **offsetTop**), so that they can be used during replay. Note that this may significantly decrease the replay speed. **Default value:** disabled.
Record 'click' by mouse events	Records mouse clicks by capturing mouse events instead of capturing the click () method. Enable when the recorded application uses the DOM click() method, to prevent the generation of multiple functions for the same user action. **Default value:** enabled.
Record socket level data	Enables the recording of socket level data. If you disable this option you will need to manually add the starting URL before recording. In addition, you will be unable to regenerate the script on an HTML level. **Default value:** enabled.
Generate snapshots for Ajax steps	Enables generation of snapshots for Ajax steps. Enabling this option can result in errors during recording. **Default value:** disabled.

Code Generation Settings Properties

UI Element	Description
Enable generation of out-of-context steps	Creates a URL-based script for ActiveX controls and Java applets, so that they will be replayed. Since these functions are not part of the native recording, they are referred to as out-of-context recording. **Default value:** disabled.
Enable automatic browser title verification	Enables automatic browser title verification. **Default value:** disabled.
Perform a title verification for	• **each navigation.** Performs a title verification only after a navigation. When a user performs several operations on the same page, such as filling out a multi-field form, the title remains the same and verification is not required. • **each step.** Performs a title verification for each step to make sure that no step modified the browser title. A modified browser title may cause the script to fail. • **Perform a title verification using the URL if the title is missing.** For browser windows without a title, perform a title verification for each step using its URL.

.NET > Recording - Recording Node

Enables you to set the recording options for .NET Vuser scripts.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Recording**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Code Generation	Allow you to indicate whether to show warnings, a stack trace, or all event subscriptions during code generation. • **Show Warnings.** Shows warning messages that are issued during the code generation process. • **Show Stack Trace.** Shows the recorded stack trace if it is available. • **Show All Event Subscriptions.** Generates code for all event subscriptions that were recorded. If this option is disabled, VuGen will only generate code for events in which both the publisher (the object which invokes the event) and the subscriber (the object informed of the event) are included in the filter. **Default value:** disabled.

Debug Options	Enables you to trace the stack and specify its size. • **Stack Trace.** Traces the contents of the stack for each invocation within the script. It allows you to determine which classes and methods were used by your application. This can be useful in determining which references, namespaces, classes, or methods to include in your filter. Enabling the trace may affect your application's performance during recording. **Default value:** disabled. • **Stack Trace Limit.** The maximum number of calls to be stored in the stack. If the number of calls exceeds the limit, VuGen truncates it. **Default value:** 20 calls.
Filters	• **Ignore all assemblies by default.** Ignores all assemblies that are not explicitly included by the selected filter. If you disable this option, VuGen looks for a matching filter rule for all assemblies loaded during the recording.
Logging	The Logging options let you set the level of detail that is recorded in the recording log file. • **Log severity.** Sets the level of logging to **Errors Only** (default), or **Debug**. The severity setting applies for all the logs that you enable below. You should always use the **Errors Only** log unless specifically instructed to do otherwise by HP support, since detailed logging may significantly increase the recording time. • **Instrumentation Log.** Logs messages related to the instrumentation process. **Default value:** enabled. • **Recording Log.** Logs messages issued during recording. **Default value:** enabled. • **Code Generation Log.** Logs messages issued during the code generation stage. **Default value:** enabled.
Remote Objects	For information about this property, see "Remote Objects Property" on next page.

Serialization	• **Serialization format.** The format of the serialization file that VuGen creates while recording a class that supports serialization: **Binary**, **XML**, or **Both**. The advantage of the binary format is that since it is more compressed, it is quicker. The disadvantage of the binary format is that you do not have the ability to manipulate the data as you do with XML. • **Serialize long arrays.** For long arrays containing serializable objects (for example, an array of primitives), use VuGen's serialization mechanism. Enabling this option generates **LrReplayUtils.GetSerializedObject** calls if the array size is equal to or larger than the threshold value. • **Threshold value for long array size.** The threshold size for an array to be considered a **long** array.If the array size is equal to or larger than this size, VuGen serializes it when detecting serializable objects. **Tip:** For XML serialization, you can view the content of the XML file. To view the file, select **View XML** from the right-click menu.

Remote Objects Property

User interface elements are described below:

UI Element	Description
Record in-process objects	Records activity between the client and server when the server is hosted in the same process as the client. Since the actions are not true client/server traffic, it is usually not of interest. When in-process methods are relevant, for example, in certain Enterprise Service applications, you can enable this option to capture them. **Default value:** disabled.
Asynchronous calls	Specifies how VuGen should handle asynchronous calls on remote objects and their callback methods • **Call original callbacks by default.** Uses the recorded application's original callback when generating and replaying the script. If the callback method is explicitly excluded by a filter, the callback will be excluded even if you enable this option. • **Generate asynchronous callbacks.** This option defines how VuGen will handle callbacks when the original callbacks are not recorded. For more information, see "Asynchronous Calls" on page 588.

WCF duplex binding	• **Generate dummy callback handler.** Replaces the original callback in duplex communication with a dummy callback, performing the following actions: ▪ **Store arguments.** When the server calls the handler during replay, it saves the method arguments to a key-value in memory map. ▪ **Synchronize replay.** It stops the script execution until the next response arrives. VuGen places the synchronization at the point that the callback occurred during recording. This is represented in the script by a warning: • **Generate unique client base address.** If your application employs dual HTTP Binding, since HTTP is inherently not a duplex protocol, the framework uses a standard port to receive response data being passed to the callback. When you attempt to run multiple instances of your application, you may be unable to do so using the same port number. This option replaces the original client base address's port number with a unique port. For background information about WCF duplex binding, see "Recording WCF Duplex Communication" on page 585.

.NET > Shared DLLs

This dialog box enables you to specify the list of shared DLLs before you record a Vuser script. If a DLL is included in the list of shared DLLs, when the Vuser script is run and requires a particular DLL, the Vuser will access the DLL in its shared location – the DLL will not be copied to the load generator. Adding a DLL to the list of shared DLLs therefore saves hard-drive space on the load generator when a Vuser is run.

Note: The location that you specify for a shared DLL must be accessible to all load generators on which the Vuser will run.

After you record a Vuser script, the list of shared DLLs is copied from the Recording Options to the Run-Time Settings. For details on how to view and modify the run-time settings, see ".NET > Shared DLLs" on page 378.

To access	**VuGen > Record > Recording Options > Microsoft .NET > Shared DLLs**
See also	".NET > Shared DLLs" on page 378

User interface elements are described below:

UI Element	Description

DLL Entries	The list of shared DLLs that VuGen will access while the Vuser script is being recorded. The order in which the DLLs appear in the list is significant. When a specific DLL is required, VuGen will access the first instance of that DLL in the list. If a specific DLL is not currently available, clear the check box to the left of the DLL entry. The DLL will remain in the list of shared DLLs. To enable the DLL, select the check box to the left of the DLL entry.
	**Down Arrow.** Moves the selected DLL entry down the list.
	**Up Arrow.** Moves the selected DLL entry up the list.
	**Add DLL.** Enables you to add a DLL to the list of shared DLLs.
	**Add DLL Folder.** This option is always disabled.
	**Delete.** Removes the selected DLL from the list of shared DLLs.

Mobile TruClient Properties > Mobile Device Node

This pane enables you to select a mobile device properties when recording a Mobile TruClient script.

To access	• **VuGen > Recording Options > Mobile TruClientProperties > Mobile Device** • Select **Moble Device** button from the Main VuGen Toolbar 
Relevant tasks	"How to Add, Remove, and Import Mobile Device Settings for Mobile TruClient" on page 582 "How to Record a Script with Mobile TruClient" on page 582

User interface elements are described below:

UI Element	Description
Mobile Device	Select the mobile device type you want to test.

UI Element	Description
User Agent	Specify the header string that is sent to server to identify your mobile device. Once you have a selected a device, the default header value will appear. However, this header string can be modified.
Display	Specify the height and width of your mobile device screen. Mobile TruClient will open browser window according to the display settings.

Network > Port Mapping Node

Enables you to set the port mapping recording options.

To access	**VuGen > Record > Recording Options > Network > Port Mapping**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Opens the Server Entry dialog box, allowing you to add a new mapping. For user interface details, see "Server Entry Dialog Box" on next page.
	Opens the Server Entry dialog box, allowing you to edit the selected entry. For user interface details, see "Server Entry Dialog Box" on next page.
	Opens the Advanced Settings dialog box to enable auto-detection of the communication protocol and SSL level. For user interface details, see "Advanced Port Mapping Settings Dialog Box" on page 316.
<Port Mapping list>	A list of the port mappings. You can temporarily disable any entry by clearing the check box adjacent to it. When you disable an entry, VuGen ignores all traffic to that server:port combination. You should disable the port entry when the data is irrelevant or if the protocol is not supported.

Capture level	The level of data to capture (relevant only for HTTP based protocols): • **Socket level data.** Capture data using trapping on the socket level only. Port mappings apply in this case (default). • **WinINet level data.** Capture data using hooks on the **WinINet.dll** API used by certain HTTP applications. The most common application that uses these hooks is Internet Explorer. Port mappings are not relevant for this level. • **Socket level and WinINet level data.** Captures data using both mechanisms. WinINet level sends information for applications that use **WinINet.dll**. Socket level sends data only if it determines that it did not originate from **WinINet.dll**. Port mapping applies to data that did not originate from **WinINet.dll**.
Network-level server address mappings for:	Specifies the mappings per protocol. For example, to show only the FTP mappings, select FTP.

Server Entry Dialog Box

Enables you to define a server from the server list in the network port mapping node.

To access	**VuGen > Record > Recording Options > Network > Port Mapping > New Entry / Edit Entry**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
See also	• Use heading and page format if cross-referencing within this portal. • Use generic and name of another portal if cross-referencing outside of this portal. • Do not use periods unless you have complete sentences.

User interface elements are described below:

UI Element	Description

Allow forwarding to target server from local port	Forwards all traffic from a specific port to another server. This is particularly useful in cases where VuGen cannot run properly on the client, such as unique Linux machines, or instances where it is impossible to launch the application server through VuGen. You configure VuGen to intercept the traffic from the problematic client machine, and pass it on to the server. In this way, VuGen can process the data and generate code for the actions. **Example:** If you were working on a Linux client called host1, which communicated with a server, server1, over port 8080, you would create a Port Mapping entry for server1, port 8080. In the Traffic Forwarding section of the Server Entry dialog box, you enable traffic forwarding by selecting the **Allow forwarding to target server from local port** check box. You specify the port from which you want to forward the traffic, in our example 8080. You then connect the client, host1, to the machine running VuGen, instead of server1. VuGen receives the communication from the client machine and forwards it via the local port 8080, to the server. Since the traffic passes through VuGen, it can analyze it and generate the appropriate code.
Connection Type	The security level of the connection: **Plain** (non-secure), **SSL**, or **Auto**. If you select Auto, the recorder checks the first 4 bytes for an SSL signature. If it detects the SSL signature, it assumes that SSL is being used.
Port	The port of the target server for which this entry applies. Entering **0** specifies all ports. If you do not specify all of the port and server names, VuGen uses the following priorities in assigning data to a service: • Priority 1: port and server specified • Priority 2: port not specified, server specified • Priority 3: port specified, server not specified • Priority 4: port and server not specified A map entry with a high priority does not get overridden by an entry with a lower priority. For example, if you specify that traffic on server twilight using port 25 be handled as SMTP and then you specify that all servers on port 25 be handled as HTTP, the data will be treated as SMTP. • **Forced mapping**. If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol to record that communication, even though the actual communication may be HTTP. In this instance, the Vuser script might be empty.
Record Type	The type of recording—directly or through a proxy server.
Service ID	A protocol or service name used by the recorder to identify the type of connection (i.e. HTTP, FTP, and so on). You can also specify a new name. The name may not exceed 8 characters.

Service Type	The type of service, currently set to TCP.
SSL Cipher	The preferred SSL cipher to use when connecting with a remote secure server.
SSL Version	The preferred SSL version to use when communicating with the client application and the server. **Default value:** SSL 2/3. However some services require SSL 3.0 only or SSL 2.0 only. Some new wireless applications require TLS 1.0—a different security algorithm.
Target Server	The IP address or host name of the target server for which this entry applies. **Default value:** All Servers.
Use specified client-side certificate	The default client-side certificate to use when connecting to a remote server. Specify or browse for a certificate file in **txt**, **crt**, or **pem** format, and supply a password.
Use specified proxy-server certificate	The default server certificate to present to client applications that request a server certificate. Specify or browse for a certificate file in **txt**, **crt**, or **pem** format, and supply a password. Click **Test SSL** to check the authentication information against the server.

Advanced Port Mapping Settings Dialog Box

Enables you to set the advanced port mapping settings. For more information, see "Port Mapping Auto Detection" on page 258.

To access	**VuGen > Record > Recording Options > Network > Port Mapping > Options**
Important information	This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Enable auto detection of SOCKET based communication	Automatically detects the type of communication. If required, raise the maximum number of transitions, one at a time until VuGen succeeds in detecting the protocol. You can also gradually increase the maximum buffer size by 1024 bytes (1 KB) at a time until VuGen succeeds in detecting the protocol. This allows VuGen to review a larger amount of data in order to find a signature.

Enable auto SSL detection	Automatically detects SSL communication. Specify the version and default cipher that you want to detect. Note that this only applies to port mappings that were defined as **auto** in the **Connection type** box, or not defined at all. If a server, port, or server:port combination was defined as either **Plain** or **SSL**, then auto SSL detection does not apply.
Log Level	Sets the logging level for the automatic socket detection.

RDP > Code Generation > Advanced Node

Enables you to control the way VuGen creates an RDP script. Only advanced users are advised to modify these settings.

To access	**VuGen > Record > Recording Options > RDP > Code Generation - Adv**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Correlate clipboard parameters	Replaces the recorded clipboard text sent by the user with the correlated parameter containing the same text as received from the server.
Double-click timeout (msec)	The maximum time (in milliseconds) between two consecutive mouse button clicks to be considered a double-click. **Default value:** 500 milliseconds.
Prefix for clipboard parameters	The prefix for clipboard parameters generated in the current script. This is useful when merging scripts, allowing you to specify a different prefix for each script. **Default value:ClipboardDataParam_**.
Prefix for snapshot names	The prefix for snapshot file names generated in the current script. This is useful when merging scripts—you can specify a different prefix for each script. **Default value: snapshot_**.

RDP > Code Generation > Agent Node

Enables you to control the way the agent for Microsoft Agent for Terminal Server functions with VuGen during recording.

To access	**VuGen > Record > Recording Options > RDP > Code Generation - Agent**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Enable RDP agent log	Enables the RDP agent log. • **RDP agent log detail level.** Configures the level of detail generated in the RDP agent log with **Standard** being the lowest level of detail and **Extended Debug** being the highest level of detail. • **RDP agent log destination.** Configures the destination of the RDP agent log data. **File** saves the log messages only on the remote server side. **Stream** sends the log messages to the VuGen machine. **FileAndStream** sends the log messages to both destinations. • **RDP agent log folder.** The folder path on the remote server that the RDP agent log file will be generated in.
Use RDP agent	Generates script using information gathered by the RDP agent during the recording session. The LoadRunner RDP agent must be installed on the server.

RDP >Code Generation > Basic Node

Enables you to control the way VuGen creates a script—the level of detail, triggers, and timeouts.

To access	**VuGen > Record > Recording Options > RDP > Code Generation - Basic**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Always generate connection name	If selected, function call will contain the **ConnectionName** parameter. If not selected, the functions will only contain this parameter if more than a single **rdp_connect_server** appears in the script. **Default value:** disabled.

Automatic generation of synchronization points	Synchronization points allow the script to pause in the replay while waiting for a window or dialog to pop-up, or some other control to fulfil a certain condition. This option automatically generates **sync_on_image** functions before mouse clicks and drags (enabled by default). The **Sync radius** is the distance from the mouse operation to the sides of the rectangle which defines the synchronization area. The default is 20 pixels. Select one of the following options: • **None.** No synchronization points are automatically added. • **Rectangular**. Creates synchronization points as rectangular boxes centered around the click or drag location. • **Enhanced**. Creates synchronization points designed to select only the desired location (e.g. a button) and to react to changes in the UI (e.g. the button moves). If a synchronization region is not recognized, the rectangular synchronization settings are used.
Generate mouse movement calls	Generates **rdp_mouse_move** calls in the script. When enabled, this option significantly increases the script size. **Default value:** disabled.
Generate raw keyboard calls	Generates **rdp_raw_key_up/down** calls as if the script level was set to **Raw**. Mouse calls will still be generated according to the script level. If disabled, VuGen generates Keyboard calls according to the script level. If the script level is set to **Raw**, this option is ignored. **Default value:** disabled.
Generate raw mouse calls	Generates **rdp_mouse_button_up/down** calls as if the script level was set to Raw. Keyboard calls will still be generated according to the script level. If disabled, VuGen generates Mouse calls according to the script level. If the script level is set to **Raw**, this option is ignored. **Default value:** disabled.
Script generation level	The level of the script and the type of API functions to use when generating the script. • **High.** Generate high level scripts. Keyboard events are translated to **rdp_type** calls. Two consecutive mouse clicks with the same coordinates are translated as a double-click. • **Low.** Generate low level scripts. Key up/down events are translated into **rdp_key** events. Modifier keys (Alt, Ctrl, Shift) are used as a **KeyModifier** parameter for other functions. Mouse up/down/ move events are translated to mouse click/drag events. • **Raw.** Generates a script on a raw level, by extracting input events from network buffers and generating calls in their simplest form: key up/down, mouse up/down/move. The **KeyModifier** parameter is not used.

RDP > Client Startup Node

Enables you to set the RDP client startup recording options.

To access	**VuGen > Record > Recording Options > RDP > Client Startup**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Run RDP client application	Connects to the terminal server by running the Terminal Services client.
Use custom connection file	Connects to the terminal server by using an existing connection file. The file should have an ***.rdp** extension. You can browse for the file on your file system or network.
Use default connection file	Connects to the terminal server by using the **Default.rdp** file in your document's folder.

Recording Properties > Corba Options Node

Enables you to set the CORBA specific recording properties and several callback options.

To access	**VuGen > Record > Recording Options > Recording Properties > Corba Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Record CallBack Connection	Instructs VuGen to generate a connect statement for the connection to the ORB, for each callback object. **Default value:** disabled.
Record DLL only	Instructs VuGen to record only on a DLL level. **Default value:** disabled.
Record Properties	Instructs VuGen to record system and custom properties related to the protocol. **Default value:** enabled.
Resolve CORBA Objects	When correlation fails to resolve a CORBA object, recreate it using its binary data. **Default value:** disabled.

Show IDL Constructs	Displays the IDL construct that is used when passed as a parameter to a CORBA invocation. **Default value:** enabled.
Use local vendor classes	Use local vendor classes and add the **srv** folder to the BOOT classpath. If you disable this option, VuGen uses network classes and adds the script's classes to the classpath. **Default value:** enabled.
Vendor	The CORBA vendors: **Inprise Visibroker**, **Iona OrbixWeb**, or **Bea Weblogic**.

Recording Properties > Correlation Options Node

Allows you to enable automatic correlation, and control its depth.

To access	**VuGen > Record > Recording Options > Recording Properties > Correlation Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Advanced Correlation	Enables correlation on complex objects such as arrays and CORBA container constructs and arrays. This type of correlation is also known as deep correlation. **Default value:** enabled.
Correlate Collection Type	Correlates objects from the Collection class for JDK 1.2 and higher. **Default value:** disabled.
Correlate String Arrays	Correlate strings within string arrays during recording. If disabled, strings within arrays are not correlated and the actual values are placed in the script. **Default value:** enabled.
Correlate Strings	Correlate strings in script during recording. If disabled, the actual recorded values are included in the script between quotation marks and all other correlation options are ignored **Default value:** disabled.
Correlation Level	Indicates the level of deep correlation, the number of inner containers to be scanned. **Default value:** 15.

Recording Properties > Log Options Node

Enables you to determine the level of debug information generated during recording.

To access	**VuGen > Record > Recording Options > Recording Properties > Log Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Class Dumping	Dumps all of the loaded classes to the script folder. **Default value:** disabled.
Digest Calculation	Generate a digest of all recorded objects. **Default value:** disabled. • **Exclude from Digest**. A list of objects not to be included in the digest calculation. **Syntax:** java.lang.Object class format, delimiter = ","
Log Level	The level of recording log to generate: • **None**. No log file is created • **Brief**. Generates a standard recording log and output redirection • **Detailed**. Generates a detailed log for methods, arguments, and return values. • **Debug.** Records hooking and recording debug information, along with all of the above.
Synchronize Threads	For multi-threaded applications, instructs VuGen to synchronize between the different threads. **Default value:** disabled.

Recording Properties > Recorder Options Node

Enables you to set the Java protocol to record as well as other protocol specific recording options.

To access	**VuGen > Record > Recording Options > Recording Properties > Recorder Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description

Byte Array Format	The format of byte arrays in a script: Regular, Unfolded Serialized Objects, or Folded Serialized Objects. Use one of the serialized object options when recording very long byte arrays. **Default value:** Regular.
Bytes as Characters	Displays readable characters as characters with the necessary casting—not in byte or hexadecimal form. **Default value:** enabled.
Comment Lines Containing	Comment out all lines in the script containing one of the specified strings. To specify multiple strings, separate the entries with commas. **Default value:** Any line with a string containing <undefined> will be commented out.
Extensions List	A comma separated list of all supported extensions. Each extension has its own hooks file. **Default value:** JNDI.
Insert Functional Check	Inserts verification code that compares the return value received during replay, to the expected return value generated during recording. This option only applies to primitive return values. **Default value:** disabled.
Load Parent Class Before Class	Change the loading order so that parent classed are loaded before child classes. This helps identify hooking for trees with deep inheritance. **Default value:** enabled.
Record LoadRunner Callback	Records the LoadRunner stub object as a callback. If disabled, VuGen records the original class as the callback. **Default value:** enabled.
Recorded Protocol	Specifies which protocol to record: RMI, CORBA, JMS, or Jacada. **Default value:** RMI.
Remove Lines Containing	Remove all lines containing one of the specified strings from the script. To specify multiple strings, separate the entries with commas. This feature is useful for customizing the script for a specific testing goal.
Unreadable Strings as Bytes	Represents strings containing unreadable characters as byte arrays. This option applies to strings that are passed as parameters to invocations. **Default value:** enabled.
Use _JAVA_ OPTIONS flag	Forces JVM versions 1.2 and higher to use the _JAVA_OPTION environment variable which contains the desired JVM parameters. **Default value:** disabled.
Use DLL hooking to attach LoadRunner support	Use DLL hooking to automatically attach LoadRunner support to any JVM.

Recording Properties > Serialization Options Node

Enables you to control how objects are serialized. Serialization is often relevant to displaying objects in an ASCII representation in order to parameterize their values.

To access	**VuGen > Record > Recording Options > Recording Properties > Serialization Options**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, "Protocol Compatibility Table" on page 264.
See also	"How to Correlate Scripts - Java Scripts - Serialization" on page 155

User interface elements are described below:

UI Element	Description
Unfold Serialized Objects	Expands serialized objects in ASCII representation and allows you to view the ASCII values of the objects in order to perform parameterization. • **Limit Object Size (bytes).** Limits serializable objects to the specified value. Objects whose size exceeds this value, will not be given ASCII representation in the script. **Default value:** 3072 bytes. • **Ignore Serialized Objects.** Lists the serialized objects not to be unfolded when encountered in the recorded script. Separate objects with commas. **Syntax:** java.lang.Object class format, delimiter = "," • **Serialization Delimiter.** Indicates the delimiter separating the elements in the ASCII representation of objects. VuGen will only parameterize strings contained within these delimiters. The default delimiter is `#'. • **Unfold Arrays.** Expands array elements of serialized objects in ASCII representation. If you disable this option and an object contains an array, the object will not be expanded. **Default value:** enabled—all deserialized objects are totally unfolded. ▪ **Limit Array Entries.** Instructs the recorder not to open arrays with more than the specified number of elements. **Default value:** 200.

RTE > Configuration Node

Enables you to set the recording options to match the character set used during terminal emulation.

To access	**VuGen > Record > Recording Options > RTE > Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Character Set	Match the character set used during terminal emulation. The default character set is ANSI. For Kanji and other multi-byte platforms, you can specify DBCS (Double-byte Character Set).

RTE > RTE Node

Enables you to set the general RTE recording options.

To access	**VuGen > Record > Recording Options > RTE > RTE**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Generate automatic synchronization commands	Automatically generates a number of **TE-synchronization** functions, and insert them into the script while you record. • **Cursor.** Generate a **TE_wait_cursor** function before each **TE_type** function. • **Prompt.** Generate a **TE_wait_text** function before each **TE_type** function (where appropriate). • **X-System.** Generate a **TE_wait_sync** function each time a new screen is displayed while recording. **Note:** VuGen generates meaningful **TE_wait_text** functions when recording VT type terminals only. Do not use automatic **TE_wait_text** function generation when recording block-mode (IBM) terminals.
Generate automatic X-System transaction	Records the time that the system was in the X SYSTEM mode during a scenario run. This is accomplished by inserting a **TE_wait_sync_transaction** function after each **TE_wait_sync** function. Each **TE_wait_sync_transaction** function creates a transaction with the name **default**. Each **TE_wait_sync_transaction** function records the time that the system spent in the previous X SYSTEM state.
Generate screen header comments	Generates screen header comments while recording a Vuser script, and inserts the comments into the script. A generated comment contains the text that appears on the first line of the terminal emulator window. **Note:** You can generate comments automatically only when using block-mode terminal emulators such as the IBM 5250.
Keyboard record timeout	When you type text into a terminal emulator while recording, VuGen monitors the text input. After each keystroke, VuGen waits up to a specified amount of time for the next key stroke. If there is no subsequent keystroke within the specified time, VuGen assumes that the command is complete.

SAPGUI > Auto Logon Node

Enables you to log on automatically when you begin recording. The logon functions are placed in the vuser_init section of the script.

To access	**VuGen > Record > Recording Options > SAPGUI > Auto Logon**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Enable Auto logon	Enables you to log on automatically when you begin recording. Enter the **Server name**, **User**, **Password**, **Client** name, and interface **Languages** for the SAP server.

SAPGUI > Code Generation Node

Enables you to set the code generation settings for the SAPGUI protocol.

To access	**VuGen > Record > Recording Options > SAPGUI > Code Generation**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Always generate Object ID in header file	Places the Object IDs in a separate header file instead of in the script. When you disable this option, VuGen generates the IDs according to the specified string length in the general script setting. This results in a more compact and cleaner script.
Generate Fill Data steps	Generates Fill Data steps for table and grid controls—instead of separate steps for each cell.
Generate logon operation as a single step	Generates a single **sapgui_logon** method for all of the logon operations. This helps simplify the code. If you encounter login problems, disable this option.

SAPGUI > General Node

Enables you to set the general recording options for the SAPGUI protocol.

To access	**VuGen > Record > Recording Options > SAPGUI > General**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.

User interface elements are described below:

UI Element	Description
Capture screen snapshots	Indicates how to save the snapshots of the SAPGUI screens as they appear during recording: **ActiveScreensnapshots**, **Regular snapshots**, or **None.** ActiveScreen snapshots provide more interactivity and screen information after recording, but they require more resources.
Changing events during recording	• **Process Context menus by text.** Processes context menus by their text, generating **sapgui_toolbar_select_context_menu_item_by_text** functions. When disabled, VuGen processes context menus by their IDs, and generates a **sapgui_toolbar_select_context_menu_item** for context menus. This is an advantage when working with Japanese characters.

Silverlight > Services Node

Enables you to manage WSDL files in Silverlight Protocol scripts.

To access	**VuGen > Record > Recording Options > Silverlight > Services**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
Relevant tasks	"How to Import WSDL Files" on page 654

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Service list>	The list of imported WSDL files and their locations. The toolbar allows you to add, delete and edit WSDL files. Additionally, select the Protocol and Security Data button to edit Protocol and Security Data.
Automatically detect WSDL files and import services during code generation	Automatically attempts to locate and import WSDL files used in your script.
Do not use WSDL files	Disables WSDL files in your script, generating SOAP requests instead. This results in a lower level script, however it generally increases the script performance.

Service Endpoint	The location of the endpoint at which a given WSDL is available.
Use WSDL files included in the script	Enables WSDL files imported automatically or manually.
WSDL Location	The location of the selected WSDL file.

Add / Edit Services Dialog Box

Enables you to locate and import WSDLs to a Silverlight Protocol script.

To access	**VuGen > Record > Recording Options > Silverlight > Services > Add**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
Relevant tasks	"How to Import WSDL Files" on page 654

User interface elements are described below:

UI Element	Description
Connection Settings	Opens the Connection Settings dialog box, enabling you to configure the proxy and authentication information for the specified WSDL file. For user interface information, see "Connection Settings Dialog Box" below.
Select WSDL from	• **URL.** Select the WSDL by specifying the URL. • **File.** Select the WSDL by specifying the local path. • **Previously Imported.** Select the WSDL from the WSDL History (list of previously imported WSDL files).
Service Endpoint	The location of the endpoint at which a given WSDL is available.
WSDL Location	The URL or local path to the WSDL.

Connection Settings Dialog Box

Configures the proxy and authentication information for WSDL files in Silverlight Protocol scripts.

To access	**VuGen > Record > Recording Options > Silverlight > Services > Add > Connection Settings**

Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • These settings are only relevant for importing WSDL files. To configure authentication and proxy information for WSDL files to be used during replay, add a **web_set_user_step** function with the desired values.
Relevant tasks	"How to Import WSDL Files" on page 654

User interface elements are described below:

UI Element	Description
Authentication	Enable the authentication settings by selecting **Use Authentication Settings** and entering your user name and password.
Proxy	Enable the proxy settings by selecting **Use Proxy Settings** and entering your user name, password, server, and port number.

Protocol and Security Scenario Data Dialog Box

Enables you to configure the protocol and security scenario data settings.

To access	**VuGen > Record > Recording Options > Silverlight > Services > Protocol and Security Data Button**
Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264. • These settings are only relevant for importing WSDL files. To configure authentication and proxy information for WSDL files to be used during replay, add a **web_set_user_step** function with the desired values. • The settings in this dialog box are reset during code generation.
Relevant tasks	"How to Import WSDL Files" on page 654

User interface elements are described below:

UI Element	Description
Port	An individual endpoint for a WSDL binding. **Note:** Selecting a different port resets the values in this dialog box to the last saved values. Any changes made that have not yet been saved will be reset.

Transport	The transport layer protocol used by VuGen to send service requests to the server. You can select HTTP, HTTPS, or LrHTTP. **Note:** HTTP is not compatible with UserNameOverTransport security and HTTPS requires that you select UserNameOverTransport security.
Encoding	The encoding method to be used for service requests sent to the server.
WS Addressing version	The WS-Addressing version for the selected WSDL file.
Security	
Authentication mode	Enable authentication by selecting **UserNameOverTransport** mode. **Default mode:** None.
Username	When authentication is enabled, a valid username is required.
Password	When authentication is enabled, a valid password is required.

Traffic Analysis > Traffic Filter

This dialog box enables you to filter either incoming or outgoing traffic.

To access	**VuGen > Record > Recording Options > Traffic Analysis > Traffic Filters**
Important information	• For details, see "Analyzing Traffic" on page 571. • This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see the "Protocol Compatibility Table" on page 264.
Relevant tasks	"How to Create a Script by Analyzing Traffic" on page 576

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
When generating the script	**Include all IP addressing the list.** Checkbox to include traffic from specified IP address. **Exclude all IP address in the list.** Checkbox to exclude traffic from specified IP addresses.
<Incoming Traffic Tab> Enables you to identify IP addresses for incoming traffic	
	Add an IP address
Source IP	Specify the IP address of the server.

UI Element	Description
	Delete IP address
<Outgoing Traffic Tab> Enables you to identify IP addresses for outgoing traffic	
	Add an IP address
Destination IP	Specify the IP address of the server.
Destination Port	Specify the destination port of the server.
	Delete IP address

WinSock Node

Enables you to set the WinSock recording options.

To access	**VuGen > Record > Recording Options > WinSock**

Important information	When using translation tables on Solaris machines, you must set the following environment variables on all machines running scripts:
	`setenv LRSDRV_SERVER_FORMAT 0025`
	`setenv LRSDRV_CLIENT_FORMAT 04e4`

Relevant tasks	"How to Record a Windows Sockets Script" on page 796

User interface elements are described below:

UI Element	Description
	Adds a new entry to the list of excluded sockets.
	Removes the selected entry from the list of excluded sockets.
Do not include excluded socket in log	Excludes the sockets on the list from the log. Clearing this option enables logging for the excluded sockets. Their actions are preceded by "Exclude" in the log file.

Exclude Settings/ Socket List	The host and port of the sockets to exclude from the recording or regeneration of the script. Use the following syntax: • **host:port** format excludes a specific port. • **host** format excludes all ports for the specified host. • **:port** format excludes a specific port on the local host. • ***:port** format excludes a specific port on all hosts.
Think Time Threshold	During recording, VuGen automatically inserts the think time steps when you pause between actions. You can set a threshold level, below which the recorded think time will be ignored. **Default value:** five seconds.
Translation tables	The Translation Table lets you specify the format for recording sessions when using the WinSock single protocol, and for code generation when using a WinSock multi protocol. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. Select a translation option from the list box. The first four digits of the listbox item represent the server format. The last four digits represent the client format. The translation tables are located in the **ebcdic** folder under the VuGen's installation folder. If your system uses different translation tables, copy them to the **ebcdic** folder. **Note:** If your data is in ASCII format, it does not require translation. Select the **None** option, the default value.

Run-Time Settings

Run-Time Settings Overview

Run-time settings define the way a Vuser script runs and can be configured before or after you record a script. These settings are stored in files located in the Vuser script folder. Run-time settings are applied to Vusers when you run a script using VuGen, the Controller,Performance Center, or Business Process Monitor.

Configuring run-time settings allows you to emulate different kinds of user activity. For example, you can emulate a user who responds immediately to output from the server, or a user who stops and thinks before each response. You can also configure the run-time settings to specify how many times the Vuser should repeat its set of actions.

You can copy run-time settings from one script to another by using copy/paste functionality for scripts in the same solution or you can export your run-time settings to a script in a different solution or on a different machine. You can also revert the run-time settings back to the default settings. For more information, see How to Work with Run-Time Settings.

Different combinations of run-time setting settings are available for each protocol. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

Content Checking Overview

You use the ContentCheck run-time options to check the contents of a page for a specific string. This is useful for detecting non-standard errors. In normal operations, when your application server fails, the browser displays a generic HTTP error page indicating the nature of the error. The standard error pages are recognized by VuGen and treated as errors, causing the script to fail. Some application servers, however, issue their own error pages that are not detected by VuGen as error pages. The page is sent by the server and it contains a formatted text string, stating that an error occurred.

For example, suppose that your application issues a custom page when an error occurs, containing the text **ASP Error**. You instruct VuGen to look for this text on all returned pages. When VuGen detects this string, it fails the replay.

Note: VuGen searches the body of the pages—not the headers.

How to Work with Run-Time Settings

This topic describes how to use the general functionality that can be used on run-time settings for all protocols. For an overview of the run-time settings functionality, see Run-Time Settings Overview.

Note: Run-time settings can only be modified with copy/paste and import/export from within VuGen.

Note: In all the tasks below:

- The script from which the run-time settings are being taken will be referred to as Script A
- The receiving script will be referred to as Script B.

How to copy run-time settings from one script to another

1. Make sure that both Script A and Script B are open.
2. In the Solution Explorer pane, right click on the Run-Time Settings node of Script A and select **Copy** in the context menu.
3. In the Solution Explorer pane, right click on the Run-Time Settings node of Script B and select **Paste** in the context menu.
4. In the message box asking if you are sure, click **Yes**.
5. Save Script B.

Note: The settings from Script A are copied to Script B using the following logic:

- Setting in Script A and setting in Script B: The setting in Script B is overwritten
- Setting in Script A and no setting in Script B: The setting in Script B is set to the value of the setting in Script A.
- No setting in Script A and setting in Script B: The setting in Script B remains the same.

How to export run-time settings from Script A

1. Open Script A.
2. In the Solution Explorer pane, right click on the Run-Time Settings node of the script and select **Export**.
3. In the Export run-time settings dialog, choose a location for the file, type a name for the file, ensure that the file type is Script Settings Files (*.xsss) and click **OK**.

How to import run-time settings to Script B

1. Open Script B.
2. In the Solution Explorer pane, right click the Run-Time Settings node of on the script and select **Import**.
3. In the Import run-time settings dialog, select the file to be imported and click **OK**.
4. In the message box asking if you are sure, click **Yes**.
5. Save Script B.

Note: The settings from Script A are copied to Script B using the following logic:

- Setting in Script A and setting in Script B: The setting in Script B is overwritten
- Setting in Script A and no setting in Script B: The setting in Script B is set to the value of the setting in Script A.
- No setting in Script A and setting in Script B: The setting in Script B remains the same.

How to revert run-time settings in a script node to the default settings

1. In the Solution Explorer pane, select the Run-Time Settings node of the script to be changed and navigate to the required setting.
2. Click **Use Defaults**.
3. Click **OK**.
4. Save the script.

Note: Only the defaults for the displayed node are changed. If you want to revert to the default settings for all the run-time settings, you must do so for each individual node separately.

Multithreading

The Controller uses a driver program (such as *mdrv.exe* or *r3vuser.exe*) to run your Vusers. If you run each Vuser as a process, then the same driver program is launched (and loaded) into the memory again and again for every instance of the Vuser. Loading the same driver program into memory uses up large amounts of RAM (random access memory) and other system resources. This limits the numbers of Vusers that can be run on any load generator.

Alternatively, if you run each Vuser as a thread, the Controller launches only one instance of the driver program (such as *mdrv.exe*), for every 50 Vusers (by default). This driver process/program launches several Vusers, each Vuser running as a thread. These threaded Vusers share segments of the memory of the parent driver process. This eliminates the need for multiple re-loading of the driver program/process saves much memory space, thereby enabling more Vusers to be run on a single load generator.

To configure these options, see "General > Miscellaneous Node" on page 359.

Protocol Compatibility Table

The following table lists the Vuser protocols and which run-time setting nodes are available for each protocol.

Protocol	Run-Time Setting Nodes
Ajax	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
Ajax TruClient	• General - Pacing, Additional Attributes, Log, Load Mode Browser Settings, Other Settings
C Vuser	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Citrix ICA	• General – Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network – Speed Simulation • Citrix – Configuration, Synchronization
COM/DCOM	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
DB2 CLI	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous

DNS	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
EJB	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Java Environment Settings - Classpath, Java VM
FTP	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
Flex	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Flex - RTMP, Configuration, Externalizable Objects • Internet Protocol - Proxy, Preferences, Download Filters, Content Check
Internet Messaging (IMAP)	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
Java over HTTP	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Java Environment Settings - Classpath, Java VM • Internet Protocol - Proxy, Preferences, Download Filters
Java Record Replay, Java Vuser	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Java Environment Settings - Classpath, Java VM
Javascript Vuser	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Internet Protocol - Proxy, Preferences, Download Filters
LDAP	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation

MMS (Media Player)	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
Microsoft.NET	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • .NET - .NET Environment
RDP	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • RDP - Configuration, Synchronization, Advanced, RDP Agent
MS Exchange (MAPI)	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
MMS (Multimedia Messaging Service)	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Browser - Browser Emulation • MMS - Server and Protocol • WAP - Radius, Gateway • Internet Protocol - Proxy, Preferences, Download Filters
ODBC	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Oracle (2-Tier)	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
Oracle NCA	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Oracle NCA - Client Emulation
Oracle Web Applications 11i	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck • Oracle NCA - Client Emulation

PeopleSoft Enterprise	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
PeopleSoft Tuxedo	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
POP3	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
Real	• General -Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
SAP Web	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SAP Click & Script	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SAP GUI	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • SAP GUI - General • Network - Speed Simulation
Siebel Web	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck

Silverlight	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Silverlight - Services • Data Format Extensions - Configuration • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck
SMTP	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation
RTE	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • RTE - RTE
Tuxedo	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous
VB Script Vuser	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Internet Protocol - Proxy, Preferences, Download Filters
VB Vuser	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Internet Protocol - Proxy, Preferences, Download Filters
Web (Click & Script)	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, ContentCheck

Web (HTTP/HTML)	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • Browser - Browser Emulation • Internet Protocol - Proxy, Preferences, Download Filters, Content Check • Data Format Extension - Configuration
Web Services	• General - Run Logic, Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation • JMS - Advanced
Windows Sockets	• General - Pacing, Log, Think Time, Additional Attributes, Miscellaneous • Network - Speed Simulation

Browser > Browser Emulation Node

Enables you to configure the browser related run-time settings.

To access	**VuGen > Replay > Run-Time Settings> Browser > Browser Emulation**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
User Agent	This displays information about the browser that is to be emulated. All Internet Vuser headers include a User Agent header that identifies the type of browser (or toolkit for Wireless) that is being emulated. For example, **Mozilla/4.0 (compatible; MSIE 6.0; Windows; Trident/4.0; English (United States))** identifies the browser as Internet Explorer running under Windows on an Intel machine. Click  to change the User Agent header. Specify the browser type, browser version, language, and operating system or enter a custom User Agent header.
Change Button	Opens the User-Agent Dialogue Box, enabling you to change the emulated browser.

Simulate Browser Cache	Instructs the Vuser to simulate a browser with a cache (enabled by default). Even if you disable this option, each resource is only downloaded once for each page, even if it appears multiple times. A resource can be an image, a frame, or another type of script file. Enabling this options allows you to set the following options: • **Cache URLs requiring content (HTML).** Instructs VuGen to cache only the URLs that require the HTML content. The content may be necessary for parsing, verification, or correlation. When you select this option, HTML content is automatically cached. **Default value:** enabled. **Tip:** To decrease the memory footprint of the virtual users, disable this option, unless it is an explicit requirement for your test. • **Check for newer versions of stored pages every visit to the page**. Instructs the browser to check for later versions of the specified URL than those stored in the cache. When you enable this option, VuGen adds the "If-modified-since" attribute to the HTTP header. This option brings up the most recent version of the page, but also generates more traffic during the scenario or session execution. **Default value:** disabled. • Advanced... . Opens the "Browser > Browser Emulation Node" on previous page, enabling you to specify the URL content types that you want to store in the cache.
Download non-HTML resources	Instructs Vusers to load graphic images when accessing a Web page during replay. This includes both graphic images that were recorded with the page, and those which were not explicitly recorded along with the page. When real users access a Web page, they wait for the images to load. Therefore, enable this option if you are trying to test the entire system, including end-user time. To increase performance and not emulate real users, disable this option. Disable this option if you experience discrepancies in image checks, since some images vary each time you access a Web page (for example, advertiser banners).
Simulate a new user each iteration	Instructs VuGen to reset all HTTP contexts between iterations to their states at the end of the init section. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. It deletes all cookies, closes all TCP connections (including keep-alive), clears the emulated browser's cache, resets the HTML frame hierarchy (frame numbering will begin from 1) and clears the user-names and passwords. **Default value:** enabled. • **Clear cache on each iteration.** Clears the browser cache for each iteration in order to simulate a user visiting a Web page for the first time. Clear the check box to disable this option and allow Vusers to use the information stored in the browser's cache, simulating a user who recently visited the page.

Cache URL Requiring Content - Advanced Dialog Box

The Advanced dialog box lets you specify the URL content types that you want to store in the cache.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Restore List	Restore the default content types if they were deleted.
	**Add.** Adds a new content type.
	**Remove.** Removes a content type.
<Content Types List>	A list of the content types along with a check box to enable/disable each type.
Specify URLs requiring content in addition to HTML page	Enables you to specify URL's.

Citrix > Configuration Node

Enables you to set the Citrix configuration run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Citrix > Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Queue mouse movements and keystrokes	Instructs Vusers to create a queue of mouse movements and keystrokes, and send them as packets to the server less frequently. This setting reduces network traffic with slow connections. Enabling this option makes the session less responsive to keyboard and mouse movements. **Default value:** disabled.
Sound quality	Specifies the quality of the sound. If the client machine does not have a 16-bit Sound Blaster-compatible sound card, select **Sound Off**. With sound support enabled, you will be able to play sound files from published applications on your client machine.
SpeedScreen Latency Reduction	The mechanism used to enhance user interaction when the network speed is slow. You can turn this mechanism **on** or **off**, depending on the network speed. The **auto** option turns it on or off based on the current network speed. If you do not know the network speed, set this option to **Use Server Default** to use the machine's default.
Use data compression	Instructs Vusers to compress the transferred data. You should enable data compression if you have a limited bandwidth. **Default value:** enabled.
Use disk cache for bitmaps	Instructs Vusers to use a local cache to store bitmaps and commonly-used graphical objects. You should enable this option if you have a limited bandwidth. **Default value:** disabled.

Citrix > Synchronization Node

Enables you to set the Citrix synchronization run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Citrix > Synchronization**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Default Image Sync Tolerance	This setting controls the level of equality two images must share to be considered synchronized. **Exact.** Must have a 100% match. **Low.** Must have a 95% match. **Medium.** Must have an 85% match. **High.** Must have a 70% match. **Default value:** Exact.
Timeout	• **Connect Time.** The time (in seconds) to wait idly at an established connection before exiting. • **Waiting Time.** The time (in seconds) to wait idly at a synchronization point before exiting.
Typing rate	The delay (in milliseconds) between keystrokes.

COM/DCOM > Filter Node

Enables you to define which COM/DCOM objects to record.

To access	Use one of the following: • **Record > Recording Options > COM/DCOM > Filter** • **Replay > Runtime Settings > COM/DCOM > Filter**

User interface elements are described below:

UI Element	Description

DCOM Profile	**Specify one of the following filter types:** • **Default Filter.** The filter to be used as the default when recording a COM Vuser script. • **New Filter.** A clean filter based on the default environment settings. Note that you must specify a name for this filter before you can record with its settings. You can also save the current settings and delete a filter using the **Save As** and **Delete** buttons.
DCOM Listener Settings List	Displays a tree hierarchy of type libraries. You can expand the tree to show all of the available classes in the type library. You can expand the class tree to show all of the interfaces supported by that class. To exclude a type library, clear the check box next to the library name. This excludes all of its classes in that type library. By expanding the tree, you can exclude individual classes or interfaces by clearing the check box next to the item. An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, a dialog box opens asking you if you also want to exclude the interface in all classes that implement it this interface. Note that when you clear the check box adjacent to an interface, it is equivalent to selecting it in the Excluded Interfaces dialog box. • **Environment.** The environments to record: ADO objects, RDS Objects, and Remote Objects. Clear the objects you do not want to record. • **Type Libraries.** A type library .**tlb** or .**dll** file, that represents the COM object to record. All COM objects have a type library that represents them. You can select a type library from the Registry, Microsoft Transaction Server, or file system. **Type Libraries.** In the lower section of the dialog box, VuGen displays the following information for each type library. • **TypLib.** The name of the type library (tlb file). • **Path.** The path of the type library. • **Guid.** The Global Unique Identifier of the type library.

Add	Adds another COM type library. • **Browse Registry.** Displays a list of type libraries found in the registry of the local computer. Select the check box next to the desired library or libraries and click **OK**. • **Browse file system**. Allows you to select type libraries from your local file system. • **Browse MTS**. add a component from a Microsoft Transaction Server. The MTS Components dialog box prompts you to enter the name of the MTS server. Type the name of the MTS server and click **Connect**. Remember that to record MTS components you need an MTS client installed on your machine. Select one or more packages of MTS components from the list of available packages and click **Add**. Once the package appears in the list of Type Libraries, you can select specific components from the package.
Remove	Removes a COM type library.
Exclude...	Excludes interfaces in the filter through the Excluded Interfaces dialog box. In this dialog box, the checked interface listings are the ones that are excluded. You can also add interfaces that are not listed. Click **Add Interface...** in the Excluded Interfaces dialog box and enter the GUID number (interface ID) and name of the interface. You can copy the GUID from the interfaces.h file created by VuGen and listed in the selection tree in the left-hand column of the VuGen screen. Use the **Add Interface...** feature to exclude interfaces that are called needlessly by the script, but are not listed anywhere in the filter. An interface can be implemented differently by various classes. When you exclude an interface that is implemented by other classes that have not been excluded, VuGen displays the a warning. If you check **Don't ask me again** and close the dialog box, then the status of all instances of the interface in all other classes will be changed automatically for this filter, whenever you change the status of the interface in one object. Click **Yes to all** to change the status of all instances of this interface for all other classes, click **No to all** to leave the status of all other instances unchanged. Click **Next Instance** to view the next class that uses this interface.

COM/DCOM > Options Node

Enables you to set additional options for your COM recording session, relating to the handling of objects, generation of logs, and VARIANT definitions.

The DCOM scripting options apply to all programming languages. These settings let you configure the scripting options for DCOM methods and interface handling.

To access	**VuGen > Replay > Run-time Settings > COM/DCOM > Options**

User interface elements are described below:

UI Element	Description
ADO Recordset filtering	Condense multiple recordset operations into a single-line fetch statement (enabled by default).
Declare Temporary VARIANTs as Globals	Define temporary VARIANT types as Globals, not as local variables (enabled by default).
Fill array in separate scopes	Fill in each array in a separate scope (enabled by default).
Fill structure in separate scopes	Fill in each structure in a separate scope (enabled by default).
Generate COM exceptions	Generate COM functions and methods that raised exceptions during recording (disabled by default).
Generate COM statistics	Generate recording time performance statistics and summary information (disabled by default).
Limit size of SafeArray log	Limit the number of elements printed in the safearray log per COM call, to 16 (enabled by default).
Release COM Objects	Record the releasing of COM objects when they are no longer in use (enabled by default).
Save Recordset content	Stores Recordset content as grids, to allow viewing of recordset in VuGen (enabled by default).
Trap binded moniker objects	Trap all of the bound moniker objects (disabled by default).

Data Format Extension > Code Generation Node

Enables Data Format Extensions during code generation.

To access	**VuGen > Replay > Run-Time Settings > Data Format Extensions > Code Generation**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 264.
Relevant tasks	• "How to Apply Chains to Sections of the HTTP Message" on page 681 • "How to use GWT-DFE" on page 682
See also	• "Data Format Extensions" on page 675 • "Data Format Extension List" on page 683

User interface elements are described below:

UI Element	Description
Enable data format extension	Enables you to select chains for each message section of the HTTP message. Deselected by default.

Flex > Configuration Node (Run-Time Settings)

Enables you to set an external JVM (Java Virtual Machine) path.

To access	**VuGen > Replay > Run-Time Settings > Flex > Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Use External JVM	Enables you to use external JVM.
External JVM Path	The path of the external JVM.

Flex > Externalizable Objects Node (Run-Time Settings)

The dialog box enables you to configure run time setting for externalizable objects in Flex scripts.

To access	**VuGen > Replay > Run-Time Settings > Flex > Externalizable Objects**
Relevant tasks	• "How to Serialize Using External Java Serializer" on page 524 • "How to Serialize Scripts with the LoadRunner Serializer" on page 525
See also	• "Flex Overview" on page 512 • " Externalizable Objects in Flex Scripts" on page 521

User interface elements are described below:

UI Element	Description
Do not serialize externalizable objects	Generate script using default settings.

Serialize objects using	Select the appropriate radio button: • Select **LoadRunner AMF serializer** if you are not using the Adobe LiveCycle Data Services or Adobe BlazeDS server. • Select **Custom Java classes** and select one or both of the options: ▪ Select **Use Flex LCDS/BlazeDS jars** if you are using Flex LCDS or BlazeDS jars to serialize the messages. If you selected **UseGraniteDS configuration** in the **Configuration** node, do not select **Use Flex LCDS/BlazeDS jars**. ▪ Select **Use additional jars** to add additional jars to serialize the messages. You must copy the jar files from the server and specify their location in the **Classpath Entries List** described below. You must ensure that the files exist in the same location on all load generator computers. If you add jars with the same names as the Flex LCDS or Blaze DS jars chosen by selecting the first check-box, these files will be overwritten.
Classpath Entries List	
	**Down Arrow.** Moves a classpath entry down the list.
	**Up Arrow.** Moves a classpath entry up the list.
	**Add Classpath.** Adds a new line to the classpath list.
	**Add Classpath Folder.** Adds all files from the folder to the classpath list.
	**Delete.** Permanently removes a classpath.

Flex > RTMP Node

Enables you to set the Flex RTMP run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Flex > RTMP**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Close all open RTMP connections after each iteration	Automatically disconnects any open RTMP connections at the end of each iteration.

TCP receive timeout (milliseconds)	Specifies the time interval in milliseconds during which waits for the client to start receiving TCP/IP packets. If a timeout has been reached, triggers a warning or error depending on the run-time setting for **Status for TCP receive timeout**.
Status for TCP receive timeout	Indicates which status to issue for a step when timeout is exceeded.
Warning	Issues a warning for the step when the timeout is exceeded. continues to the next step.
Error	Issues an error for the step when the timeout is exceeded. The script ends.

General > Additional Attributes Node

You can use the Additional Attributes node to provide additional arguments for a Vuser script. The Additional Attributes settings apply to all Vuser script types.

You specify command line arguments that you can retrieve at a later point during the test run, using **lr_get_attrib_string**. Using this node, you can pass external parameters to prepared scripts.

To access	**VuGen > Replay > Run-Time Settings > General > Additional Attributes**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Additional Attributes table>	A list of the additional attributes and their values.
Add	Add a new argument.
Remove	Remove an argument.

General > Log Node

Enables you to configure the amount and types of information that is recorded in the log.

To access	**VuGen > Replay > Run-Time Settings > General > Log**

User interface elements are described below:

UI Element	Description

Enable logging	Enables automatic logging during replay. Disabling this options only affects automatic logging and log messages issued through **lr_log_message**. Messages sent manually, using **lr_message**, **lr_output_message**, and **lr_error_message**, are still issued.
Log options	Indicates when messages are sent to the log: • **Send messages only when an error occurs**. Sends messages to the log only when an error occurs. Click **Advanced** to configure the log cache size. If the contents of the cache exceed the specified size, VuGen deletes the oldest items. • **Always send messages.** Sends all messages to the log.
Standard log	Creates a standard log of functions and messages sent during script execution to use for debugging. You can disable this option for large load testing scenarios or profiles if you wish to save system resources.
Extended log	Creates an extended log, including warnings and other messages. You can disable this option for large load testing scenarios or profiles if you wish to save system resources. You can also specify the following options: • **Parameter substitution.** Logs all parameters assigned to the script along with their values. • **Data returned by server.** Logs all of the data returned by the server. • **Advanced trace.** Logs all of the functions and messages sent by the Vuser during the session.

General > Log Node (AjaxTruClient)

Enables you to configure the amount and types of information that is reported to a log for Ajax TruClient scripts.

To access	**VuGen > Replay > Run-Time Settings > General > Log**

User interface elements are described below:

UI Element	Description

Log Level and Associated Options	**Error-only** • Select to log only error messages. **Standard log** • Log errors/warnings only (in Standard log). Select to log errors and warnings only. Informative messages are not included. In Extended log, all messages ignored due to this option are included, regardless of this setting. **Extended Log** • Select to log low-level informative messages and all messages included in the Standard log. **HTTP-related messages** • **Log HTTP request headers.** Log HTTP request header of each request. • **Log HTTP request body (POST data).** Log the HTTP request body of each request. • **Log HTTP response headers**. Log the HTTP response headers of each response. • **Log HTTP response body.** Log the HTTP response body of each response. • **Log HTTP trace.** Log the HTTP request/response handling Application under Test (AUT) messages **Application under test (AUT) messages** • **Log AUT error messages.** Logs error messages which are received from the application under test. • **Log AUT non-error messages.** Log informative messages which are received from the application under test. **Parameterization** • **Log Parameter substitution.** Logs the parameters and the values used when the Vuser script runs.
Create a log file only if required	**Accumulate messages in memory, write only if required.** • Select to log messages only when an error occurs. • **Accumulate Messages in Memory Buffer Size.** Enter the memory buffer size where important messages are logged. **Note:** If the size is too small, messages will be lost. If the size is too large, this may contribute to paging issues and slow execution.

Log Message Formatting Options	**User Log Line Length.** • Enter the limit for a single line in a user log file before the message line begins to wrap. **Include Timestamps in user log.** • Select to include time trace to each message in the log.
Internal Support Options	These options are for use with a customer support representative only. Do no modify them unless specifically instructed to do so by customer support.

 General > Advanced Settings (TruClient)

This dialog box enables you to configure advanced settings for Mobile TruClient.

To access	**VuGen > Replay > Run Time Settings > General > Advanced Settings**

User interface elements are described below:

UI Element	Description
Enable Plugin Support	If selected, enables plug in support, such as Flash.

Load > Browser Settings Node (Ajax TruClient)

This node enables you to configure settings for the Ajax TruClient browser for scripts that you run in load mode.

To access	**VuGen > Replay > Run Time Settings > Load > Browser Settings**
Important information	• Settings that you modify in the Load Mode Browser Settings node affect load mode only. • The settings available in the Load Mode Browser Settings node are the same settings that are available in the" Ajax TruClient General Settings Dialog Box" on page 405. ▪ Settings that you modify in the Browser Settings tab affect interactive mode only. When you save your script in interactive mode, any settings that you modified in the Browser Settings tab are applied to the Load Mode Browser Settings.

 FireFox Browser Settings

User interface elements are described below:

UI Element	Description

Proxy selection	
No proxy (direct connection to the internet)	For all Vusers, make the connection to the Internet directly without using a proxy server.
Use system proxy settings	(Default) Use system configuration for proxy settings.
Manual proxy configuration	Enables you to set and configure proxy server settings.
Do not use a proxy for (exceptions)	You can specify exceptions to the proxy server rules. You can list the exceptions as hostnames, IPs and/or masks. **Note:** You can use wildcards as part of the hostname, e.g. *.devlab.ad.
Host for HTTP/all protocols	The domain name or IP address of the proxy server used forHTTP and other protocol requests.
Port for HTTP/all protocols	The port number of the proxy server used for HTTP and other protocol requests.
Use separate proxy for HTTPS protocol	Enables you to use a separate proxy server to handle all HTTPS protocol requests.
Host for HTTPS protocol	The domain name or IP address of the proxy server used for HTTPS protocol requests.
Port for HTTPS protocol	The port number of the proxy server used for HTTPS protocol requests.
AUTOMATIC proxy configuration (PAC)	Automatically read the proxy settings from the proxy auto configuration file (PAC). In the **URL** field, enter the URL of the PAC file.
Advanced	
Home Page URL	Enter the URL to navigate to when window home is called from JavaScript. (about:blank by default)
User-Agent	Enter the User-Agent string for overriding the browser default in request headers.
Compare the page in cache to the page on the network	Enter the corresponding number used to compare the URL page in the cache to the URL page called from the network. • **0.** (Default) Once per session. • **1.** Every time the page is accessed. • **2.** Never. • **3.** When the page is out of date (default).

Keep-Alive	Allows persistent (non-proxied) network connections. • **Checked.** (Default)Allow persistent (non-proxied) network connections, so that the open connections can be reused. • **Cleared.** Close each connection after the request is complete.
Proxy Keep-Alive	Allows persistent proxy connections. • **Checked.** (Defeat) Allow persistent proxy connections, so that open connections can be reused. • **Cleared.** Close each connection after the request is complete.
Keep-Alive timeout (sec)	Enter the number of seconds to keep idle connections open. (300 by default)
HTTP version	Enter the HTTP version to use when accessing the network/application not via a proxy. (1.1 by default)
Proxy HTTP version	Enter the Proxy HTTP version to use when accessing the network/application via a proxy. (1.1 by default)
DNS cache entries	Enter the maximum number of entries to keep in the DNS cache. Enter 0 to disable this option. (20 by default)
SSL	Select the secure connection setting/s: • SSL 2.0 • (Default) SSL 3.0 • (Default) TLS 1.0

Load > Other Settings (TruClient)

This dialog box enables you to configure snapshot generation and action on error for TruClient.

To access	**VuGen > Replay > Run Time Settings > Load > Other Settings**

User interface elements are described below:

UI Element	Description
Snapshot Generation	
Replay snapshots generation	**Never.** If selected, snapshots are never generated on replay. **On Error.** If selected, snapshots are generated on steps with errors during replay. **Always.** If selected, snapshots are generated on all steps in the script during replay.

UI Element	Description
Action on error	**Abort script.** Select to abort running script when an error occurs. **Continue to the next iteration.** Select to stop current iteration and continue to the next one.

General > Replay Node

All Protocols - Except Mobile Ajax TruClient

To access	**VuGen > Replay > Run-Time Settings > General > Replay**
UI Element	**Description**
Simulate a new user on each iteration	Instructs VuGen to reset all HTTP contexts between iterations. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. **Default value:**enabled.
Proxy selection	The settings in this section apply to both recording and running the script. They apply to this script only. To configure proxy settings for all Ajax TruClient scripts, open the **Browser Configuration Settings** dialog box. For more information, see " Ajax TruClient General Settings Dialog Box" on page 405. **Retrieve global proxy settings (defined in Ajax TruClient Browser Options).** Select to use the proxy settings defined in the Browser Options dialog box. The settings are retrieved when you click Develop Script. Once global proxy settings are retrieved, they are copied to the script's run-time settings and can be managed in the "Define proxy settings for script" section. Define specific proxy settings for the script. • No proxy (direct connection to the internet). For all Vusers, make the connection to the Internet directly without using a proxy server. • Manual proxy configuration. If selected, you can specify exceptions to the proxy server rules, specify the proxy server for all HTTP/HTTPS connections. you require a separate proxy server for HTTP connections, check Use separate proxy for HTTPS protocol box and specify the proxy server settings. • AUTOMATIC proxy configuration (PAC). If selected, the proxy settings is automatically read from the proxy auto configuration file (PAC). In the URL field, enter the URL of the PAC file.

Snapshots generation	**Recording snapshots generation** • Never. If selected, recording snapshots of the script are not automatically saved. • Always. If selected, recording snapshots are automatically saved for every step in the script. **Replay snapshots generation** • Never. If selected replay snapshots are not generated. **Note:** This is the default setting. • On error. Select to generate a snapshot when errors occurs. This option can be used to help identify bugs in your script. For more information, see Debug Scripts Using Snapshots. • Always. If selected snapshots are generated for every step in the script.
Action on error	Abort script Select to abort running script when an error occurs. Continue to the next iteration Select to stop iteration when an error occurs and continue to the next one.
Replay Options	Maximum time for object-not-found (seconds) Enter the maximum time to search for an object before the application displays an error message. Inter-step interval (milliseconds) Enter the minimal interval between steps. Specifying a higher value can help with synchronization issues, but too high a value may slow the script more than necessary. End-of-network identification timeout (milliseconds) The end-of-network for a step is recognized when the specified time has elapsed with no networking activity.

Advanced	A number of options for advanced users. For more details, read the tips displayed at the bottom of the Run-Time settings dialog box: • **Home Page URL.** Enter the URL to navigate to when window home is called from JavaScript. • **User-Agent.** Enter the User-Agent string for overriding the browser default in request headers. • **Compare the page in cache to the page on the network.** Enter the corresponding number used to compare the URL page in the cache to the URL page called from the network. 0=Once per session, 1=Every time the page is accessed, 2=Never, 3=When the page is out of date (default). • **Keep-Alive.** Select to allow persistent (non-proxied) network connections. Enter the following corresponding number: 1=Allow persistent (non-proxied) network connections, so that the open connections can be reused, 0=Close each connection after the request is complete. • **Proxy Keep-Alive.** Select to allow persistent proxy connections. Enter the following corresponding number: 1=Allow persistent proxy connections, so that open connections can be reused. 0=Close each connection after the request is complete. • **Keep-Alive timeout (sec).** Enter the number of seconds to keep idle connections open. • **HTTP version.** Enter the HTTP version to use when accessing the network/application not via a proxy. • **Proxy HTTP version.** Enter the Proxy HTTP version to use when accessing the network/application via a proxy. • **DNS cache entries.** Enter the max number of entries to keep in the DNS cache. **Note:** Enter 0 to disable this option. • **Non-interactive window size.** Enter the initial dimensions (width and height in pixels) of the browser window in non-interactive mode. • **SSL.** Select the secure connection setting/s: ▪ SSL 2.0 ▪ SSL 3.0 ▪ TLS 1.0

Mobile Ajax TruClient

UI Element	Description

Simulate a new user on each iteration	Instructs VuGen to reset all HTTP contexts between iterations. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. **Default value:**enabled.
Replay Options	
Replay using recorded duration for steps	During replay, each step's minimum time of running will be as recorded. **Note:** After editing the script, this might cause false timing of steps in the script.
Failed event causes step failure	If the **End Event** of a step fails, the step will fail. For example, if the **End Event** of a step is **Document Loaded** and the document fails to load, the step will fail. This option is enabled by default.
Enable profiling	Enable/disable profiling for client side breakdown. **Maximum time for object-not-found (seconds)** Enter the maximum time to search for an object before the application displays an error message. **Step timeout** The amount of time to perform a step. If the End Event does not occur b this time the step fails. Inter-step interval (milliseconds) Enter the minimal interval between steps. Specifying a higher value can help with synchronization issues, but too high a value may slow the script more than necessary. **End-of-network identification timeout (milliseconds)** The end-of-network for a step is recognized when the specified time has elapsed with no networking activity. **Typing interval (milliseconds)** The average amount of time between keystrokes when entering text during replay.

Minimum Time	Random step's minimum time within range. During replay, each step's minimum time settings will be overridden by a randomized value. **From (percentage).** Set lower percentage. **To (percentage).** Set upper percentage. **Limit step's minimum time.** Set the step's minimum time value in seconds. **Apply minimum time setting on wait steps.** Enable/disable applying minimum time setting on wait steps.
Automatic transactions	**No automatic transactions.** Do not apply automatic transactions. **Define each action as a transaction.**. Treat each action in the script as a transaction for performance measurements. **Define each action as step as a transaction.** Treat each action and step in as a transaction for performance measurements.

General > Miscellaneous Node

Enables you to set miscellaneous run-time settings.

To access	**VuGen > Replay > Run-Time Settings > General > Miscellaneous**
Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335. • We do not recommend enabling both the **Continue on Error** and **Generate Snapshot on Error** options in a load test environment. This configuration may adversely affect the Vusers' performance. • The following protocols should not be run as threads: Sybase-Ctlib, Sybase-Dblib, Tuxedo, and PeopleSoft-Tuxedo. • If you require the Vusers to generate breakdown data for diagnostics (J2EE) during the scenario run, do not use automatic transactions. Instead, manually define the beginning and end of each transaction.

User interface elements are described below:

UI Element	Description

Automatic Transactions	• **Define each action as a transaction.** Instructs LoadRunner (not applicable to HP Business Service Management) to handle every action in the script as a transaction. • **Define each step as a transaction.** Instructs LoadRunner (not applicable to HP Business Service Management) to handle every step in the script as a transaction.
Error Handling	• **Continue on Error.** Continue script execution when an error occurs. **Default value:** disabled. • **Fail open transactions on lr_error_message.** Instructs VuGen to mark all transactions in which an **lr_error_message** function was issued, as *Failed*. The **lr_error_message** function is issued through a manually defined *If* statement. • **Generate Snapshot on Error**. Generates a snapshot when an error occurs. You can see the snapshot by viewing the Vuser Log and double-clicking on the line at which the error occurred.
Multithreading	• **Run Vuser as a process.** Disables multithreading. • **Run Vuser as a thread.** Enables multithreading. For details, see "Multithreading" on page 335.

General > Pacing Node

Allows you to control the time between iterations. The pace tells the Vuser how long to wait between iterations of your actions.

To access	**VuGen > Replay > Run-Time Settings > General > Pacing**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
After the previous iteration ends	Starts each new iteration a specified amount of time after the end of the previous iteration. Specify either an exact number of seconds or a range of time.
As soon as the previous iteration ends	The new iteration begins as soon as possible after the previous iteration ends.
At <fixed/random> intervals	You specify the time between iteration—either a fixed number of seconds or a range of seconds from the beginning of the previous iteration. Each scheduled iteration will only begin when the previous iteration is complete.

General > Run Logic Node

Enables you to set the run logic run-time settings.

To access	**VuGen > Replay > Run-Time Settings > General > Run Logic**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.
See also	"Script Sections" on page 106

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Insert Action	Inserts a new Action at the insertion point.
Insert Block	Inserts a new Action block at the insertion point.
Delete	Deletes an item.
Move Up	Moves an item up.
Move Down	Moves an item down.
Properties...	Opens the Properties Dialog Box enabling you to set the run logic and iterations settings. **Run Logic**. Configures the action to run sequentially or randomly. **Iterations**. Sets the number of times an item will run.
<Run logic tree>	A graphical illustration of the run logic for this script.
Number of Iterations	The number of times the script will run of the items in the run logic tree.

General > Think Time Node

Enables you to configure the think time settings, controlling the time that a VuGen waits between actions. These settings are designed to help you emulate a real user.

To access	**VuGen > Replay > Run-Time Settings > General > Think Time**
See also	• VuGen uses **lr_think_time** functions to record think time values into your Vuser scripts. For more information about the lr_think_time function and how to modify it manually, see the Function Reference (Help > Function Reference).

User interface elements are described below:

UI Element	Description
As recorded	During replay, use the argument that appears in the lr_think_time function. For example, `lr_think_time(10)` waits ten seconds.
Ignore think time	Ignore the recorded think time—replay the script ignoring all **lr_think_time** functions.
Limit think time to	Limit the think time's maximum value.
Multiply recorded think time by	During replay, use a multiple of the recorded think time. This can increase or decrease the think time applied during playback. For example, if a think time of four seconds was recorded, you can instruct your Vuser to multiply that value by two, for a total of eight seconds. To reduce the think time to two seconds, multiply the recorded time by 0.5.
Replay the think time	Enables options which let you customize the recorded think times.
Use random percentage of recorded think time	Use a random percentage of the recorded think time. You set a range for the think time value by specifying a range for the think time. For example, if the think time argument is 4, and you specify a minimum of 50% and a maximum of 150%, the lowest think time can be two (50%) and the highest value six (150%).

Internet Protocol > ContentCheck Node

Enables you to check websites for content during run-time.

To access	**VuGen > Replay > Run-Time Settings > Internet Protocol > ContentCheck**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Delete	Deletes the selected rule or application.
Export...	Exports rules to an xml file.
Import...	Imports rules from an existing xml file.
New Application	Adds a new application to the list of applications and rules. Click to change the name.

New Rule	Displays the rule criteria in the right pane, allowing you to enter a new rule for the currently selected application. • **Search for text.** The text of the string for which you want to search. • **Search by prefix and suffix.** The prefix and suffix of the string for which you want to search. • **Match case.** Perform a case sensitive search. • **Fail if.** Configure the results of the search to fail if the string is either found or not found. • **Search JavaScript alert box text.** Only search for text within JavaScript alert boxes (Web , PeopleSoft Enterprise, and Oracle Web Applications 11i Vusers only).
Set as Default...	Set the ContentCheck configuration as default (i.e. new scripts will start with this configuration).
<Application and Rule list>	A list of applications and rules. You can enable and disable individual items by using the check boxes to the left of each item.
Enable ContentCheck during replay	Enable content checking during replay. Note that even after you define applications, you can disable it for a specific test run, by disabling this option. **Default value:** enabled.

Internet Protocol > Download Filters Node

Enables you to set the download filters.

To access	**VuGen > Replay > Run-Time Settings > Internet Protocol > Download Filters**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Filter list>	A list of filters for the script. Each filter has a type and data. For example, a filter of type URL would have a URL address as its data. You can **Add**, **Edit**, **Remove**, or **Remove All** entries in the list.
Exclude addresses in list	Ignore requests from the listed sites or hosts.
Include only addresses in list	Restrict replay to the listed sites or hosts.

Internet Protocol > Preferences Node

Enables you to set various internet related run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Internet Protocol > Preferences**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Checks	• **Enable Image and Text Check.** Allows the Vuser to perform verification checks during replay by executing the verification functions **web_find** or **web_image_check**. This option only applies to statements recorded in HTML-based mode. Vusers running with verification checks use more memory than Vusers who do not perform checks. **Default value:** disabled.
Generate Web Performance Graphs	Instructs a Vuser to collect data used to create Web Performance graphs. You view the **Hits per Second**, **Pages per Second**, and **Response Bytes per Second** (Throughput) graphs during test execution using the online monitors and after test execution using the Analysis. You view the Component Breakdown graph after test execution using the Analysis. Select the types of graph data for the Vuser to collect. If you do not use the Web performance graphs, disable these options to save memory.

Advanced	• **WinInet Replay.** Instructs VuGen to use the WinInet replay engine instead of the standard Sockets replay. VuGen has two HTTP replay engines: Sockets-based (default) or WinInet based. The WinInet is the engine used by Internet Explorer and it supports all of the features incorporated into the IE browser. The limitations of the WinInet replay engine are that it is not scalable and does not support Linux. In addition, when working with threads, the WinInet engine does not accurately emulate the modem speed and number of connections. VuGen's proprietary sockets-based replay is a lighter engine that is scalable for load testing. It is also accurate when working with threads. The limitation of the sockets-based engine is that it does not support SOCKS proxy. If you are recording in that type of environment, use the WinInet replay engine. **Default value:** disabled (socket-based replay engine). • **File and line in automatic transaction names.** Creates unique transaction names for automatic transactions by adding file name and line number to the transaction name. **Default value:** enabled. • **Non-critical item errors as warnings.** This option returns a warning status for a function which failed on an item that is not critical for load testing, such as an image or Java applet that failed to download. This option is enabled by default. If you want a certain warning to be considered an error and fail your test, you can disable this option. You can set a content-type to be critical by adding it to the list of Non-Resources. For more information, see "Non-Resources Dialog Box" on page 301. • **Save snapshot resources locally.** Instructs VuGen to save the snapshot resources to files on the local machine. •  . Opens the "Advanced Options Dialog Box" below.

Advanced Options Dialog Box

Enables you to set the advanced internet preference run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Internet Protocol > Preferences > Options**
Important information	• This dialog box is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335. • This dialog box divides the properties into different categories: **HTTP**, **General**, **Authentication**, **Log**, and **Web (Click & Script) Specific**.

HTTP

User interface elements are described below:

UI Element	Description
HTTP version	Specifies which version HTTP to use: version 1.0 or 1.1. This information is included in the HTTP request header whenever a Vuser sends a request to a Web server. **Default value:** HTTP 1.1. HTTP 1.1 supports the following features: • Persistent Connections—see "Keep-Alive HTTP connections" below. • HTML compression—see "Accept Server-Side Compression" on next page below. • Virtual Hosting—multiple domain names sharing the same IP address.
Keep-Alive HTTP connections	Keep-alive is a term used for an HTTP extension that allows persistent or continuous connections. These long-lived HTTP sessions allow multiple requests to be sent over the same TCP connection. This improves the performance of the Web server and clients. The keep-alive option works only with Web servers that support keep-alive connections. This setting specifies that all Vusers that run the Vuser script have keep-alive HTTP connections enabled. **Default value:** enabled.
Accept-Language request header	Provides a comma-separated list of accepted languages. For example, **en-us**, **fr**, and so forth. For more details, see "Page Request Header Language" on page 813.
HTTP errors as warnings	Issues a warning instead of an error upon failing to download resources due to an HTTP error.
HTTP-request connect timeout (seconds)	The time, in seconds, that a Vuser will wait for the connection of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. Note that this timeout also applies to the time the Vuser will wait for a WAP connection, initiated by the **wap_connect** function. **Default value:** 120 seconds.
HTTP-request receive timeout (seconds)	The time, in seconds, that a Vuser will wait to receive the response of a specific HTTP request within a step before aborting. Timeouts provide an opportunity for the server to stabilize and respond to the user. **Default value:** 120 seconds.
Request Zlib Headers	Sends request data to the server with the **zlib** compression library headers. By default, requests sent to the server include the **zlib** headers. This option lets you emulate non-browser applications that do not include **zlib** headers in their requests. To exclude these headers, set this option to **No**. **Default value:** Yes.

, continued

Accept Server-Side Compression	Indicate to the server that the replay can accept compressed data. The available options are: **None** (no compression), **gzip** (accept gzip compression), **gzip, deflate** (accept gzip or deflate compression), and **deflate** (accept deflate compression). Note that by accepting compressed data, you may significantly increase the CPU consumption. Default value: Accept **gzip** and **deflate** compression. To manually add compression, enter the following function at the beginning of the script: `web_add_auto_header("Accept-Encoding", "gzip");` To verify that the server sent compressed data, search for the string Content-Encoding: gzip in the section of the server's responses of the replay log. The log also shows the data size before and after decompression.
Delete unreferenced cache entries	Delete cache entries that have not been referenced within the specified number of iterations. Set to zero (0) to never delete cache entries.

General

User interface elements are described below:

UI Element	Description
DNS caching	Instructs the Vuser to save a host's IP addresses to a cache after resolving its value from the Domain Name Server. This saves time in subsequent calls to the same server. In situations where the IP address changes, as with certain load balancing techniques, be sure to disable this option to prevent Vuser from using the value in the cache. **Default value:** enabled.
Convert from/to UTF-8	Converts received HTML pages and submitted data from and to UTF-8. You enable UTF-8 support in the recording options. For more information, see "Recording Options" on page 258. **Default value:** no.
Step timeout caused by resources is a warning	Issues a warning instead of an error when a timeout occurs due to a resource that did not load within the timeout interval. For non-resources, VuGen issues an error. **Default value:** disabled.
Parse HTML Content-Type	When expecting HTML, parse the response only when it is the specified content-type: **HTML**, **text\html**, **TEXT** any text, or **ANY**, any content-type. Note that text/xml is not parsed as HTML. **Default value: TEXT**. The timeout settings are primarily for advanced users who have determined that acceptable timeout values should be different for their environment. The default settings should be sufficient in most cases. If the server does not respond in a reasonable amount of time, check for other connection-related issues, rather than setting a very long timeout which could cause the scripts to wait unnecessarily.

Step download timeout (sec)	The time that the Vuser will wait before aborting a step in the script. This option can be used to emulate a user behavior of not waiting for more than x seconds for a page.
Network buffer size	Sets the maximum size of the buffer used to receive the HTTP response. If the size of the data is larger than the specified size, the server will send the data in chunks, increasing the overhead of the system. When running multiple Vusers from the Controller, every Vuser uses its own network buffer. This setting is primarily for advanced users who have determined that the network buffer size may affect their script's performance. The default is 12K bytes. The maximum size is 0x7FFF FFFF.
Print NTLM information	Print information about the NTLM handshake to the standard log.
Print SSL information	Print information about the SSL handshake to the standard log.
Max number of error matches issued as ERRORS	Limits the number of error matches issued as ERRORS for content checks using a LB or RB (left boundary or right boundary). This applies to matches where a failure occurs when the string is found (Fail=Found). All subsequent matches are listed as informational messages. **Default value:** 10 matches.
Maximum number of META Refresh to the same page	The maximum number of times that a META refresh can be performed per page. **Default value:** 2.
ContentCheck values in UTF-8	Store the values in the ContentCheck XML file in UTF-8.
Tree-View request body limit	Limit the number of request body bytes displayed in Tree-View. Set to zero (0) for no limit.
IP Version	The IP version to be used: IPv4, IPv6 or automatic selection. The default value is IPv4.
web-sync retry interval (millisec)	The time to wait (in milliseconds) between testing the condition that yields false and the next retry. The default value is 1000.
web-sync retry timeout (millisec)	The maximum time (in milliseconds) during which retries are allowed. If the computed timeout exceeds the step timeout (as determined by the 'Step download timeout' setting), the latter is used.

Authentication

User interface elements are described below:

UI Element	Description
Fixed think time upon authentication retry (msec)	Automatically adds a think time to the Vuser script for emulating a user entering authentication information (username and password). This think time will be included in the transaction time. **Default value:** 0.
Disable NTLM2 session security	Use full NTLM 2 handshake security instead of the more basic NTLM 2 session security response. **Default value:** No.
Use Windows native NTLM implementation	Use the Microsoft Security API for NTLM authentication instead of the indigenous one. **Default value:** No.
Override credentials in Windows native NTML implementation	Use the credentials provided by the user at logon.
Enable integrated Authentication	Enable Kerberos-based authentication. When the server proposes authentication schemes, use **Negotiate** in preference to other schemes. **Default value:** No.
Induce heavy KDC load	Do not reuse credentials obtained in previous iterations. Enabling this setting will increase the load on the KDC (Key Distribution Server). To lower the load on the server, set this option to **Yes** in order to reuse the credentials obtained in previous iterations. This option is only relevant when Kerberos authentication is used. **Default value:** No.

Log

User interface elements are described below:

UI Element	Description
Print buffer line length	Line length for printing request/response header/body and/or JavaScript source, disabling wrapping.
Print buffer escape only binary zeros	• **Yes.** Escape only binary zeros when printing request/response headers/body and/or JavaScript source. • **No**. Escape any unprintable/control characters.

Web (Click & Script) Specific

User interface elements are described below:

UI Element	Description

, continued

General	• **Home Page URL.** The URL of the home page that opens with your browser (default is about:blank). • **DOM-based snapshots.** Instructs VuGen to generate snapshots from the DOM instead of from the server responses. **Default value:** Yes. • **Charset conversions by HTTP.** Perform charset conversions by the `Content-Type:....; charset=...' HTTP response header. Overrides `Convert from /to UTF-8.' • **Reparse when META changes charset.** Reparse HTML when a META tag changes the charset. Effective only when **Charset conversions by HTTP** is enabled. **Auto** means reparsing is enabled only if it used in the first iteration. • **Fail on JavaScript error.** Fails the Vuser when a JavaScript evaluation error occurs. **Default value:** No (issue a warning message only after a JavaScript error, but continue to run the script). • **Initialize standard classes for each new window project.** When enabled, the script—the src compiled script, will not be cached. • **Ignore acted on element being disabled.** Ignore the element acted on by a Vuser script function being disabled.
Timers	• **Optimize timers at end of step.** When possible, executes a setTimeout/setInterval/<META refresh> that expires at the end of the step before the expiration time **Default value:** Yes. • **Single setTimeout/setInterval threshold (seconds).** Specifies an upper timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, these methods will not invoke the functions that are passed to them. This emulates a user waiting a specified time before clicking on the next element. **Default value:** 5 seconds. • **Accumulative setTimeout/setInterval threshold (seconds).** Specifies a timeout for the window.setTimeout and window.setInterval methods. If the delay exceeds this timeout, additional calls to window.setTimeout and window.setInterval will be ignored. The timeout is accumulative per step. **Default value:** 30 seconds. • **Reestablish setInterval at end of step.0** = No; **1** = Once; **2** = Yes. • **Limit no-network timers at end of step:** Limit the number of setTimeout/setInterval specified script evaluations at the end of a step when no network requests are issued. Set to zero (0) for no limit. The default value is 100. This limit is only used when 'Optimize timers at end of step' is enabled.

, continued

History	• **History support.** Enables support for the window.history object for the test run. The options are **Enabled**, **Disabled**, and **Auto**. The **Auto** option instructs Vusers to support the window.history object only if it was used in the first iteration. Note that by disabling this option, you improve performance. **Default value:** auto. • **Maximum history size.** The maximum number of steps to keep in the history list. **Default value:** 100 steps.
Navigator Properties	• **navigator.browserLanguage.** The browser language set in the navigator DOM object's **browserLanguage** property. **Default value:** The recorded value. Scripts created with older recording engines use **en-us** by default. • **navigator.systemLanguage.** The system language set in the navigator DOM object's **systemLanguage** property. **Default value:** The recorded value. Scripts created with older recording engines use **en-us** by default. • **navigator.userLanguage.** The user language set in the navigator DOM object's **userLanguage** property. **Default value:** The recorded value. Scripts created with older recording engines use **en-us** by default.
Screen Properties	• **screen.width** Sets the width property of the screen DOM object in pixels. **Default value:** 1024 pixels. • **screen.height** Sets the height property of the screen DOM object in pixels. **Default value:** 768 pixels. • **screen.availWidth** Sets the availWidth property of the screen DOM object in pixels. **Default value:** 1024 pixels. • **screen.availHeight.** Sets the availHeight property of the screen DOM object in pixels. **Default value:** 768 pixels.

, continued

Memory Management	• **Default block size for DOM memory allocations.** Sets the default block size for DOM memory allocations. If the value is too small, it may result in extra calls to malloc, slowing the execution times. Too large a block size, may result in an unnecessarily big footprint. **Default value:** 16384 bytes. • **Memory Manager for dynamically-created DOM objects.Yes**—Use the Memory Manager for dynamically-created DOM objects. **No—**Do not use the Memory Manager, for example when multiple DOM objects are dynamically created in the same document as under SAP. **Auto**—Use the protocol recommended (default Yes for all protocols except for SAP). • **JavaScript Runtime memory size (KB).** Specifies the size of the JavaScript runtime memory in kilobytes. **Default value:** 256 KB. • **JavaScript Stack memory size (KB).** Specifies the size of the JavaScript stack memory in kilobytes. **Default value:** 32 KB.
Web Javascript	• **Enable running Javascript code**. **Yes**—Enables running web Javascript steps, such as `web_js_run()` and `web_js_reset()`. **No**—Web Javascript steps can not be run. Note that enabling this option causes the creation of a Javascript Engine Runtime, even if there are no Javascript steps in the script. **Default value:** No • **Javascript Engine runtime size (KB)**. Specifies the size of the Javascript Engine Runtime memory in kilobytes. One Runtime will be created for all Vusers in a process. **Default value:** 10240 KB • **Javascript Engine stack size per-thread (KB)**. Specifies the size of each Vuser thread in the Javascript Engine memory in kilobytes. **Default value:** 32 KB

Internet Protocol > Proxy Node

Enables you to set the proxy server connection settings.

To access	**VuGen > Replay > Run-Time Settings > Internet Protocol > Proxy**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
No proxy	All Vusers should use direct connections to the Internet. This means that the connection is made without using a proxy server.
Obtain the proxy settings from the default browser	All Vusers use the proxy settings of the default browser from the machine upon which they are running.
Use custom proxy	All Vusers use a custom proxy server. You can supply the actual proxy server details or the path of a proxy automatic configuration script (. pac file) that enables automatic configuration. • **Use automatic configuration script.** Allows you to specify a JavaScript file containing proxy assignment information. This script tells the browser when to access a proxy server and when to connect directly to the site, depending on the URL. In addition, it can instruct the browser to use a specific proxy server for certain addresses and another server for other addresses. Specify the location of the script in the **Address** field. • Use proxy server. You can specify one proxy server for all **HTTP** sites, and another proxy server for all **HTTPS** (secure) sites or check the **use the same proxy server for all protocols box**. •  . Allows you to specify exceptions to the proxy server rules. •  . Opens the Proxy Authentication Dialog Box. If the proxy server requires authentication for each Vuser, use this dialog box to enter the relevant password and user name. To add authentication dynamically during recording, or to add authentication for multiple proxy servers, use the web_set_user function. For more information, see the Function Reference (Help > Function Reference).

Java Environment Settings > Java Classpath Node

The **ClassPath** section lets you specify the location of additional classes that were not included in the system's classpath environment variable. You may need these classes to run Java applications and insure proper replay.

You can browse for the required classes on your computer or network and disable them for a specific test. You can also manipulate the classpath entries by changing their order.

To access	**VuGen > Replay > Run-Time Settings > Java Environment Settings > Classpath**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
	**Add Classpath.** Adds a new line to the classpath list.
	**Delete.** Permanently removes a classpath.
	**Down Arrow.** Moves a classpath entry down the list.
	**Up Arrow.** Moves a classpath entry up the list.
Classpath Entries List	A list of classpath entries.

Java Environment Settings > Java VM Node

Enables you to set the Java VM run-time settings.

To access	**VuGen > Replay > Run-Time Settings > Java Environment Settings > Java VM**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Virtual Machine Settings	• **Use internal logic to locate JDK.** Search the PATH, registry, and Windows folder for the JDK to use during replay. • **Use specified JDK.** Use a specified JDK during replay. • **JDK.** The home folder of the specified JDK. • **Additional VM Parameters.** Any optional parameters used by the virtual machine. • **Using Xbootclasspath parameters.** Replays the script with the Xbootclasspath /p option.
Class Loading Settings	**Load each Vuser using dedicated class loader.** Load each Vuser using a dedicated class loader. This will allow you to use a unique namespace for each Vuser and manage their resources separately.

JMS > Advanced Node

Enables you to set the JMS Advanced run-time settings.

To access	**VuGen > Replay > Run-Time Settings > JMS > Advanced**

Important information	This node is available only for the Web Service protocol. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.
Relevant tasks	"How to Prepare Scripts for Replay" on page 726

User interface elements are described below:

VM (Virtual Machine)

UI Element	Description
Use external VM	Enables you to select a VM (Virtual Machine) other than the standard one. If you disable this option, Vusers use the JVM provided with VuGen.
JVM Home	Location of the external JVM. This should point to the JDK home folder, defined by JDK_HOME. VuGen supports JDK 1.4 and above.
Classpath	Vendor implementation of JMS classes together with any other required supporting classes, as determined by the JMS implementation vendor.

JMS

UI Element	Description
Additional VM Parameters	Extra parameters to send to the JVM such as Xbootclasspath, and any parameters specified by the JVM documentation.
JNDI initial context factory	The fully qualified class name of the factory class that will create an initial context. Select a context factory from the list or provide your own.
JNDI provider URL	URL string of the service provider. For example: • `Weblogic - t3://myserver:myport` • `Websphere - iiop://myserver:myport`
JMS connection factory	JNDI name of the JMS connection factory. You can only specify one connection factory per script.
JMS security principal	Identity of the principal (for example the user) for the authentication scheme.
JMS security credentials	The principal's credentials for the authentication scheme.

Number of JMS connections per process	The number of JMS connections per mdrv process, or Vuser. All Vusers sharing a connection will receive the same messages. The default is 1, and the maximum is 50 Vusers. The fewer connections you have per process, the better your performance.
Received message timeout options	The timeout for received messages: • **Infinite wait.** Wait as long as required for the message before continuing. • **No wait.** Do not wait for the Receive message, and return control to the script immediately. If there was no message in the queue, the operation fails. • **Specify the timeout in seconds.** A timeout value for the message. If the timeout expired and no message has arrived, the operation fails. (default) • **User defined timeout.** The amount of seconds to wait for the message before timing out. The default is twenty seconds. **Default value:** No wait.
Automatically generate selector	Generates a selector for the response message with the correlation ID of the request (**No** by default). Each JMS message sent to the server has a specific ID. Enable this option if you want VuGen to automatically create a selector that includes the message ID.

MMS > Server and Protocol Node

Enables you to set the MMS (Multimedia Messaging Service) run-time settings.

To access	**VuGen > Replay > Run-Time Settings > MMS > Server and Protocol**
Important information	• This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335. • In the **General > Miscellaneous node,** under Multithreading, select **Run Vuser as a process.**

User interface elements are described below:

UI Element	Description
Automatic WAP Connections	Defines when to connect and disconnect from a WAP gateway. This setting is only relevant when a WAP gateway is used. The possible values are: **Per Iteration.** Connect at the beginning of each iteration and disconnect at the end of each iteration. **Per Send or Receive.** Connect and disconnect at the beginning and end of each message. **None.** Do not use automatic WAP connections. **Default value:** Per Iteration.

Default Sender address	The default address sent in the Sender header. **Default value:** +999999.
MMS Version	The version of the MMS protocol used by the script.
MMSC URL	The URL of the MMSC (Multimedia Messaging Center) server.
SMSC IP	The IP address of the SMSC server used for sending MMS notifications over SMPP.
SMSC Port	The IP port of the SMSC server used for sending MMS notifications over SMPP.
Timeout (seconds)	The time that the server waits for incoming messages. **Default value:** 60 seconds.

Mobile TruClient Properties > Mobile Device Node

This pane enables you to select a mobile device properties when recording a Mobile TruClient script.

To access	• **VuGen > Replay > Run Time Settings > Mobile TruClient Properties > Mobile Device Node** • Select **Moble Device** button from the Main VuGen Toolbar 
Relevant tasks	"How to Add, Remove, and Import Mobile Device Settings for Mobile TruClient" on page 582 "How to Record a Script with Mobile TruClient" on page 582

User interface elements are described below:

UI Element	Description
Mobile Device	Select the mobile device type you want to test.
User Agent	Specify the header string that is sent to server to identify your mobile device. Once you have a selected a device, the default header value will appear. However, this header string can be modified.
Display	Specify the height and width of your mobile device screen. Mobile TruClient will open browser window according to the display settings.

.NET > .NET Environment Node

Enables you to set the run-time settings for .NET Vuser scripts.

To access	**VuGen > Replay > Run-Time Settings > .NET > .NET Environment**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
AUT configuration	Configuration settings for the AUT location and configuration files. • **AUT Application Base Path.** The AUT (Application Under Test) base folder from which DLLs are loaded during replay. By default, during recording, all of the necessary DLLs are stored in the script's folder. Use this option to specify the location of any missing DLL files for the AUT. This is usually the installation path of the recorded application. Note that the AUT must be installed on the machine running the script. If you leave this box empty, VuGen uses the local script\bin folder as the application base folder during replay. • **AUT Configuration File.** The file name of the recorded application's configuration file. VuGen copies the AUT configuration file to the **script\bin** folder and loads the locally saved file. To specify a different location, enter a full path. If you only specify a file name, and the file is not in the **script\bin** folder, VuGen loads it from the App base folder.
Concurrency	**AppDomain Per Vuser.** Enables execution of each Vuser in a separate app domain. Running Vusers in separate App Domains enables each Vuser to execute separately without sharing static variables and prevents locking between them. **Default value:** true.

.NET > Shared DLLs

This dialog box enables you to view and modify the list of shared DLLs after you record a Vuser script. If a DLL is included in the list of shared DLLs, when the Vuser script is run and requires a particular DLL, the Vuser will access the DLL in its shared location – the DLL will not be copied to the load generator. Adding a DLL to the list of shared DLLs therefore saves hard-drive space on the load generator when a Vuser is run.

Note: The location that you specify for a shared DLL must be accessible to all load generators on which the Vuser will run.

After you record a Vuser script, the list of shared DLLs is copied from the Recording Options to the Run-Time Settings. For details on how to view and modify the recording options, see ".NET > Shared DLLs" on page 311.

To access	VuGen > Replay > Run-Time Settings > Microsoft .NET > Shared DLLs
See also	".NET > Shared DLLs" on page 311

User interface elements are described below:

UI Element	Description
DLL Entries	The list of shared DLLs that the Vusers will access when the Vuser script is run.
	**Down Arrow.** Moves the selected DLL entry down the list.
	**Up Arrow.** Moves the selected DLL entry up the list.
	**Add DLL.** Enables you to add a DLL to the list of shared DLLs.
	**Add DLL Folder.** This option is always disabled.
	**Delete.** Removes the selected DLL from the list of shared DLLs.

Network > Speed Simulation Node

Enables you to configure the network speed run-time settings.

To access	VuGen > Replay > Run-Time Settings > Network > Speed Simulation
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Use bandwidth	Indicate a specific bandwidth level for your Vuser to emulate. You can select a speed ranging from 14.4 to 512 Kbps, emulating analog modems, ISDN, or DSL.
Use custom bandwidth	Indicate a bandwidth limit for your Vuser to emulate. Specify the bandwidth in bits, where 1 Kilobit=1024 bits.
Use maximum bandwidth	Vusers run at the maximum bandwidth that is available over the network. **Default value:** enabled.

Oracle NCA > Client Emulation Node

Enables you to set the Oracle NCA run-time settings.

To access	VuGen > Replay > Run-Time Settings > Oracle NCA > Client Emulation

Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Network	• **Socket Mode.** Configure the connection between the client and server at the socket level. ▪ **Timeout.** The time that an Oracle NCA Vuser waits for a response from the server. **Default value:** -1 (disables the timeout and the client waits indefinitely). • **Pragma Mode.** Configure the communication on the Oracle-defined Pragma mode (higher level than socket and HTTP). ▪ **Max Retries.** The maximum number of **IfError** messages the client will accept from the server before issuing an error. **IfError** messages are the periodic messages the server sends to the client, indicating that it will respond with the data as soon as it is able. ▪ **Retry Interval (ms).** The interval (in milliseconds) between retries in the case of **IfError** messages. ▪ **Include retry intervals in transaction.** Includes the interval between retry time, as part of the transaction duration time. • **Enable Heartbeat.** Sends a heartbeat signal is sent to the server. You can configure the frequency of the heartbeat by setting the **frequency** property. **Default value:** enabled, 120.
Connection	• **Forms version.** The version of the Oracle Forms server detected during recording. Modify this setting only if the server was upgraded since the recording. • **Command line parameter separator.** Marker for the division between separate parameters in a command line string.
Diagnostic	• **Application Version.** The version of Oracle Application. This option is relevant when using Oracle Application—not a custom Oracle NCA application. It is only required when using Oracle database breakdown.

RDP > Advanced Node

Enables you to set the RDP advanced run-time settings.

To access	**VuGen > Replay > Run-Time Settings > RDP > Advanced**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Bitmap caching	Allows the remote desktop server to use bitmap caching. Enabling this setting can save system resources on the remote desktop server.
Font smoothing	Allows the remote desktop server to use font smoothing. Disabling this setting can save system resources on the remote desktop server.
Menu and window animation	Allows the remote desktop server to animate menus and windows. Disabling this setting can save system resources on the remote desktop server.
Remote desktop composition	Enables remote desktop composition.
Show contents of window while dragging	Shows the contents of windows while they are being dragged. Disabling this setting can save system resources on the remote desktop server.
Show remote desktop background image	Allows you to run the remote desktop application without displaying the desktop background image on the remote desktop. Disabling this setting can save system resources on the remote desktop server.
Socket receive buffer size (bytes)	The number of bytes to allocate for the socket's receive buffer. If the buffer is too small, it can fill up causing the server to disconnect. If the buffer is too large, it uses more local system resources (memory).
Themes	Allows the remote desktop server to use Windows themes. Disabling this setting can save system resources on the remote desktop server.

RDP > Configuration Node

Enables you to set the RDP configuration run-time settings.

To access	**VuGen > Replay > Run-Time Settings > RDP > Configuration**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Enable RDP caching	Support data caching orders in RDP (enabled by default).
RDP Client Version Emulation	The version of RDP packets to produce during replay: **As Recorded**, or a specific version number.

Remote desktop color depth	The color depth settings for the replay: **As Recorded**, or a specific depth.
Remote desktop resolution (pixels)	The size of the window in which the applications are run: **As Recorded**, or a specific size (in pixels).
Start the following program on connection	Open RDP connection to invoke the specified application. Specify the following information: **Program path and file name** and optionally, **Start in folder**.

RDP > RDP Agent Node

Enables you to set the RDP Agent run-time settings.

To access	**VuGen > Replay > Run-Time Settings > RDP > RDP Agent**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Use RDP agent	Instructs VuGen to use RDP agent during recording, then generates script using information gathered by the RDP agent during the recorded session. LoadRunner RDP agent must be installed on the server.
Enable RDP agent log	Enables the RDP agent log. This feature is should be used only for debugging purposes. • **RDP agent log detail level.** Configures the level of detail generated in the RDP agent log with **Standard** being the lowest level of detail and **Extended Debug** being the highest level of detail. • **RDP agent log destination.** Configures the destination of the RDP agent log data. **File** saves the log messages only on the remote server side. **Stream** sends the log messages to the VuGen machine. **FileAndStream** sends the log messages to both destinations. • **RDP agent log folder.** The folder path on the remote server that the RDP agent log file will be generated in. If none is specified and the agent log destination was set to **File**, the log is saved in the temp folder of the user on the server.

RDP > Synchronization Node

Enables you to set the RDP synchronization run-time settings.

To access	**VuGen > Replay > Run-Time Settings > RDP > Synchronization**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Default input origin	The default origin for input operations.
Default offset addition	Saves the offset of images that moved during synchronization for all subsequent functions. **Default value:** No.
Default synchronization timeout (sec)	The time in seconds to wait for synchronization operations. Enter a value between 0 and 1000. **Default value:** 60.
Default tolerance for image synchronization	The tolerance level for performing synchronization on images. Select one of the options: **Exact**, **Low**, **Medium**, or **High**. **High** has the most tolerance for changes and mismatches. **Low** requires a match of approximately 95 percent, **Medium** requires a match of approximately 85 percent, **High** requires a match of approximately 70 percent, and **Exact** requires an 100 percent match. **Default value:** Medium.
Disable synchronization failure dialog	When selected, it prevents the Synchronization Failure Dialog box from opening. **Default value:** not selected.
Fail image synchronization step on timeout	Instructs Vusers how to proceed when images are not found during synchronization. **Yes** sets a Fail status and Vusers follow the Continue on Error setting. **No** returns an LR_NOT_FOUND flag, the step reports a warning and the script continues. **Default value:** Yes.
Recorded	Uses coordinates for all input operations with a non-specified input origin. **Default value:** enabled.
Synched	Adds the most recent offsets saved at one of the previous synchronization functions to the recorded coordinates of each input operation with a non-specified input origin.
Typing speed (msec/char)	The time in milliseconds for sending consecutive characters in keyboard commands. Enter a value between 0 and 1000. **Default value:** 150.

RTE > RTE Node

Enables you to set the RTE run-time settings.

To access	**VuGen > Replay > Run-Time Settings > RTE > RTE**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Maximum number of connection attempts	The TE_connect function is generated by VuGen when you record a connection to a host. When you replay an RTE Vuser script, the TE_connect function connects the terminal emulator to the specified host. If the first attempt to connect is not successful, the Vuser retries a number of times to connect successfully. **Default value:** 5.
Use original device name	In certain environments, each session (Vuser) requires a unique device name. The TE_connect function generates a unique 8-character device name for each Vuser, and connects using this name. Select this option to connect using the device name that is contained within the com_string parameter of the TE_connect function. **Note:** The original device name setting applies to IBM block-mode terminals only.
Delay before typing	The delay setting determines how Vusers execute **TE_type** functions. • **First key.** Specify the amount of time (in milliseconds) that a Vuser waits before entering the first character in a string. • **Subsequent keys.** Specifies the amount of time (milliseconds) that a Vuser waits before between submitting successive characters. **Note:** You can use the **TE_typing_style** function to override the Delay settings for a portion of a Vuser script.
X-System synchronization	• **Timeout.** The timeout (in seconds) to wait for the system to stabilize when replaying a **TE_wait_sync** function before an error is returned. • **Stable time.** The time (in milliseconds) that the Vuser waits to ensure that the terminal is no longer in the X-SYSTEM mode after executing a **TE_wait_sync** function.

SAPGUI > General Node

Enables you to set the SAP GUI run-time settings.

To access	**VuGen > Replay > Run-Time Settings > SAPGUI > General**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Send status bar text	Send the text from the status bar to the log file.

Send active window title	Send the active window title text to the log file.
Show SAP Client during replay	Shows an animation of the actions in the SAP client during replay. The benefit of displaying the user interface (UI) is that you can see how the forms are filled out and closely follow the actions of the Vuser. This option, however, requires additional resources and may affect the performance of your load test. • **Take ActiveScreen snapshots during replay.** Captures replay snapshots with the Control ID information for all active objects. ActiveScreen snapshots differ from regular ones, in that they allow you to see which objects were recognized by VuGen in the SAP GUI client. As you move your mouse across the snapshot, VuGen highlights the detected objects. You can then add new steps to the script directly from within the snapshot. It also allows you to add steps interactively from within the snapshot for a specific object. For more information, see "How to Enhance SAP GUI Scripts" on page 645.
Options...	Opens the SAP GUI Advanced Options Dialog Box, enabling you to set the following settings: • **Replay using running SAPlogon application.** Instructs the Vusers to use the SAPlogon application that is currently running for replay. • **Set SAPfewgsvr application timeout.** Allows you to modify the **SAPfewgsvr.exe** process timeout. ▪ **Timeout to SAPfewgsvr.** The SAPfewgsvr.exe process timeout in seconds. **Default value:** 300 seconds.

Silverlight > Services Node

Displays the WSDL files associated with your script and allows you to modify their settings for the replay phase.

To access	**VuGen > Replay > Run-Time Settings > Silverlight > Services**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Service List>	A list of the WSDL files that are available for this script.
Protocol =; Security Data	Opens the Protocol and Security Scenario Data dialog box, allowing you to configure a number of settings for each selected WSDL. For more information, see "Protocol and Security Scenario Data Dialog Box" on page 329.

WAP > Gateway Node

Enables you to set the WAP gateway run-time settings.

To access	**VuGen > Replay > Run-Time Settings > WAP > Gateway**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
HTTP Direct	Run the Vusers run in HTTP mode, accessing a Web server directly.
WAP Gateway Property	Run the Vusers accessing a Web server via a WAP Gateway. • **IP.** The IP of the gateway. • **Port.** The port of the gateway. When running your Vusers through a WAP gateway, VuGen automatically sets default port numbers, depending on the selected mode. However, you can customize the settings and specify a custom IP address and port for the gateway. • **WAP 1.x (WSP).** Selects the appropriate WAP version. If you recorded in WAP 1.x (WSP), you can run the Vuser in either 1.x (WSP), or 2.0 (HTTP proxy) mode. If you select this option, you can set the WAP 1.x (WSP) Properties. For more information, see "WAP 1.x (WSP) Properties" below. • **WAP 2.0 (HTTP).** Selects the appropriate WAP version. If you recorded in WAP 2.0 (HTTP proxy), then you can only run the Vuser in the same mode.

WAP 1.x (WSP) Properties

UI Element	Description
Advanced	Expand to set the Advanced Properties. For more information see "Advanced Properties" on next page.
Connection Options	• **Connection-oriented Mode** sets the connection mode for the WSP session to Connection-Oriented. • **Connectionless Mode** sets the connection mode for the WSP session to Connectionless.
Enable Security	Enable a secure connection to the WAP gateway.

Advanced Properties

UI Element	Description
Acknowledge headers	Returns standard headers that provide information to the gateway. **Default value:** Disabled.
BearerType	The type of bearer used as the underlying transport.
CAPSessionResume	Enables requests for session suspend or resume.
Client SDU buffer size	The largest transaction service data unit that may be sent to the client during the session. **Default value:** 4000.
Confirm Push support	In CO mode, if a push message is received, this option instructs the Vuser to confirm the receipt of the message. For more information, see "VuGen Push Support" on page 803.
MethodMOR	The number of outstanding methods that can occur simultaneously.
Network MTU Size	The maximum size in bytes, of the network packet. **Default value:** 4096.
Push support	Enables push type messages across the gateway. **Default value:** Disabled.
PushMOR	The number of outstanding push transactions that can occur simultaneously.
Retrieve messages	When a push messages is received, this option instructs the Vuser to retrieve the message data from the URL indicated in the push message. **Default value:** Disabled.
Server SDU buffer size	The largest transaction service data unit that may be sent to the server during the session. **Default value:** 4000.
Support Cookies	Provide support for saving and retrieving cookies. **Default value:** Disabled.
WTLS Abbreviated Handshake	Use an abbreviated handshake instead of a full one, when receiving a redirect message. **Default value:** False.
WTLS Deffie Hellman	Use the Deffie Hellman encryption scheme for WTLS (Wireless Transport Layer Security) instead of the default scheme, RSA. **Default value:** False.
WTLS Deffie Hellman identifier	An identifier for the Deffie Hellman encryption scheme. This identifier is required for the abbreviated handshake with the Operwave gateway that uses the Deffie Hellman encryption scheme.

WTP Retransmission Time	The time in seconds that the WTP layer waits before re-sending the PDU if it did not receive a response. **Default value:** 5000.
WTP Segmentation and Reassembly	Enables segmentation and reassembly (SAR) in WTP, Wireless Transport Protocol. **Default value:** True.

WAP > Radius Node

Enables you to set the WAP radius run-time settings.

To access	**VuGen > Replay > Run-Time Settings > WAP > Radius**
Important information	This node is available only for specific protocols. For a complete list of protocols and their associated nodes, see "Protocol Compatibility Table" on page 335.

User interface elements are described below:

UI Element	Description
Accounting port number	Accounting port of the Radius server.
Authentication port number	Authentication port of the Radius server.
Connection Timeout (sec)	The time in seconds to wait for the Radius server to respond. **Default value:** 120 seconds.
IP Address	IP address of the Radius server.
Network Type	Accounting network type: GPRS (General Packet Radio Service) or CSD (Circuit-Switched Data).
Radius client IP	Radius packets source IP, usually used to differentiate between packets transmitted on different NIC cards on a single Load Generator machine.
Retransmission retries	The number of times to retry after a failed transmission. **Default value:** 0.
Secret Key	The secret key of the Radius server.
Store attributes returned by the server to parameters	Allow Vusers to save attributes returned by the server as parameters, which can be used at a later time. **Default value:** False.

Protocols

Ajax - Click & Script Protocol

Ajax - Click & Script - Protocol Overview

Ajax (Asynchronous JavaScript and XML) represents a group of technologies for creating interactive Web applications. With Ajax, web pages exchange small packets of data with the server, instead of reloading an entire page. This reduces the amount of time that a user needs to wait when requesting data. It also increases the interactive capabilities and enhances the usability.

Using Ajax, developers can create fast Web pages using Javascript and asynchronous server requests. The requests can originate from user actions, timer events, or other predefined triggers.

Ajax components, also known as Ajax controls, are GUI based controls that use the Ajax technique—they send a request to the server when a trigger occurs.

For example, a popular Ajax control is a **Reorder List** control that lets you drag components to a desired position in a list. VuGen's support for Ajax implementation is based on Microsoft's ASP.NET Ajax Control Toolkit formerly known as Atlas.

Ajax - Click & Script - Supported Frameworks

The supported frameworks for Ajax Click & Script functions are:

- Atlas 1.0.10920.0/ASP.NET Ajax—All controls
- Scriptaculous 1.8—Autocomplete, Reorder List, and Slider

VuGen supports the following frameworks at the engine level. This implies that VuGen will create standard Web (Click & Script) steps, but not Ajax specific functions:

- Prototype 1.6
- Google Web Toolkit (GWT) 1.4

Ajax - Click & Script - Example Script

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported Ajax controls, VuGen generates a function with an **ajax_xxx** prefix.

In the following example, a user selected item number 1 (index=1) in an Accordion control. VuGen generated an **ajax_accordion** function.

```
web_browser("Accordion.aspx",
                DESCRIPTION,
                ACTION,
                "Navigate=http://labm1app08/AJAX/Accordion/.aspx",
                LAST);
lr_think_time(5);

ajax_accordion("Accordion",
                  DESCRIPTION,
                 "Framework=atlas",
                 "ID=ctl00_SampleContent_MyAccordion",
```

```
                    ACTION,
                  "UserAction=SelectIndex",
                  "Index=1",
                  LAST);
web_edit_field("free_text_2",
        "Snapshot=t18.inf",
        DESCRIPTION,
        "Type=text",
        "Name=free_text",
        ACTION,
        "SetValue=FILE_PATH",
        LAST);
```

Note: When you record an Ajax session, VuGen generates standard Web (Click & Script) functions for objects that are not one of the supported Ajax controls. In the example above, the word FILE_PATH was typed into an edit box.

Click & Script - Recording Tips

This section lists tips for recording click-and-script Vuser scripts.

Use the Mouse and not the Keyboard

It is preferable to click on an object with the mouse rather than using the keyboard. During recording, use only GUI objects that are within the browser's pane. Do not use any browser icons, controls, the Stop button, or menu items, such as **View > Refresh**. You may, however, use the Refresh, Home, Back and Forward buttons and the address bar.

Do not Record Over an Existing Script

It is best to record into a newly created script—not an existing one.

Avoid Context Menus

Avoid using context menus during recording. Context menus are right-click menus which pop up when clicking certain objects in a graphical user interface.

Avoid Working in Another Browser While Recording

While recording, do not work in any browser window other than the browser windows opened by VuGen.

Wait for Downloads

Wait for all downloads to complete before doing any action, such as clicking on a button or filling in a text field.

Wait for Pages to Load

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Navigate to the Start Page

If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page at the end of your recording, so that the same link will be visible at the end of the business process.

Use a Higher Event Configuration Level

Record the business process again using the **High** event configuration level. For more information on changing the event configuration level, see "Click and Script - Troubleshooting and Limitations" on page 666.

Disable Socket Level Recording

In certain cases, the capturing of the socket level messages disrupts the application. For most recordings, socket level data is not required. To prevent the recording of socket level data, disable the option in the recording options. For more information, see "GUI Properties > Advanced Node" on page 307.

Enable the "Record rendering-related property values" Option

If the client-side scripts of the application use a lot of styling activities, enable the **Record rendering-related property values** option before recording the script. For example, enable this option to record additional DOM properties such as **offsetTop**. Note that enabling this option may decrease the recording speed. You can enable the option by selecting **Recording Options > GUI Properties > Advanced**. For more information, see "GUI Properties > Advanced Node" on page 307.

Click & Script - Replay Tips

This section lists tips for replaying click-and-script Vuser scripts.

Do not Reorder Statements in a Recorded Script

Do not change the order of the statements within a recorded script. Also, copying segments of code from one Action to another is not recommended.

Convert non-ASCII Characters

If your links contain non-ASCII characters, you should instruct VuGen to convert the data to or from the UTF-8 format.

Enable UTF-8 Conversion

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate the **Convert from/to UTF-8** option and set it to **Yes**.

Alternatively, view the list of options that is displayed when a link is not found. Enter the displayed text as-is, such as the hex escape sequences \xA0 or any other non-standard format.

Run the Same Sequence of Actions Twice

In some cases, you can perform a certain process only once—such as deleting a user from the

database. Replay will fail after the first iteration because the action is no longer valid. Verify that your business process can be repeated more than once with the same data.

Set Unique Image Properties

In the Step Navigator, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you can add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the Function Reference (**Help > Function Reference**).

Check the Step's Description

If you receive a **GUI Object is not found** error, check the Output pane for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the script in the Editor and manually modify the value of the step's DESCRIPTION argument.
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument. For more information, see the Function Reference (**Help > Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property. For more information, see the Function Reference (**Help > Function Reference**).

Click & Script - Miscellaneous Tips

The following additional tips may help you in troubleshooting problems that you experience with click-and-script Vuser scripts:

Search for Warnings

Search for warnings or alerts in the Output pane.

Verify the Response

Verify the response of the previous step is correct using **web_reg_find**. For more information, see the Function Reference (**Help > Function Reference)**.

Use Alternate Navigation

For problematic steps or those using Java applets, use **Alternative Navigation** to replace the Web step with an HTTP level step. Note that the HTTP level steps may require manual correlations. To perform Alternative Navigation, select a step in the **Step Navigator**, or the text in Script View, and select **Replace with alternative navigation** from the right-click menu.

Working with the Kerberos Protocol

If you are using the Kerberos Protocol for authentication, you must customize VuGen to properly convene authorization sessions. Advanced users can attempt to perform this customization

themselves.

In order for the Kerberos Protocol to work properly, create a krb5.ini file and put it in an available folder. Save the full path name of krb5.ini into the KRB5_CONFIG environment variable.

The krb5.ini file should contain detailed information about each domain (KDS and AS addresses) and trust chains.

For more information, contact HP software support.

Click and Script - Troubleshooting and Limitations

This section describes troubleshooting and limitations for click-and-script protocols. Some items apply to specific click-and-script protocols only.

Recording Issues and Limitations

Firefox is not supported

Only Internet Explorer is supported for Web (Click & Script). To record browser activity on Firefox, use the Web (HTTP/HTML) protocol.

Application behaves differently while being recorded

If your application behaves differently during recording, than it does without recording, you should determine if the recording problem is unique to Web. The effect may be that a Web page will not load, part of the content is missing, a popup window does not open, and so forth.

Create a new Web (HTTP/HTML) script and repeat the recording.

In the event that the recording fails in Web (HTTP/HTML), we recommend that you disable socket level recording (see "Click & Script - Recording Tips" on page 663).

The problem may be the result of an event listener. Use trial and error to disable event listeners in the **Web Event Configuration** Recording Options, and then re-record your session as a Web user.

Certain Click and Script steps do not generate properly

After recording a script, if not all steps are correctly generated, the problem may be due to the **Windows Component > Internet Explorer Enhanced Security Configuration**.

Remove **Internet Explorer Enhanced Security Configuration** by selecting **Control Panel > Add or Remove Programs > Add or Remove Windows Components** and re-record your script.

Disable an Event Listener

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Recording > Recording Options** and select the **GUI Properties > Web Event Configuration** node.
3. Click **Custom Settings** and expand the **Web Objects** node. Select an object.
4. Select **Disabled** from the list in the **Record** column for the relevant Web object. If the recording still does not work, enable the listener you previously disabled, and try disabling another one. Repeat these steps until your recording succeeds.

Dynamic menu navigation was not recorded

A dynamic menu is a menu that dynamically changes depending on where you select it. If the dynamic menu navigation was not recorded, record again using "high" event configuration mode. These settings can be found in the **Recording Options > GUI Properties > Web Event Configuration** node.

Certain user actions were not recorded

Check if there is a Java applet running inside the browser. If not, record the script with the Web (HTTP/HTML) protocol.

Replay Issues

GUI object not found

Does the error occur at the beginning of the second iteration?

If the error occurs at the beginning of the second iteration's Action section, it is probably the result of a starting page that was present for the first iteration, but missing for the second one. If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page, so that the same link will be visible at the end of the business process.

Is it a text link containing non-ASCII characters?

If the problem occurs with non-ASCII characters, you should instruct VuGen to covert the data to a suitable character set.

Enable Data Conversion on Windows Machines

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate **Charset Conversions by HTTP** in the Web (Click & Script) > General options, and set it to **Yes**.

Enable UTF-8 conversion for Linux Machines

Click **Record > Recording Options** to open the Recording Options dialog box.

Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.

Click **Options** to open the Advanced Options dialog box.

Locate **Convert from/to UTF-8** in the General options and set it to **Yes**.

Alternatively, view the list of alternatives that are displayed when a link is not found. Enter the displayed text as-is, such as hex escape sequences \xA0 or any other non-standard format.

Can you run the same sequence of actions twice in the application?

In some cases, you can only perform a certain process once, such as deleting a user from the database. Replay will fail after the first iteration, because the action is no longer valid. Verify that your business process can be repeated in the application more than once with the same data, without recording again.

Were the image properties 'Id', 'Name' and 'Alt' empty?

In the **Step Navigator**, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the Function Reference (**Help > Function Reference**).

Did the step's description change?

Check the Output pane for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the Script View and manually modify the value of the step's DESCRIPTION argument(s).
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument(s). For more information, see the Function Reference (**Help > Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property For more information, see the Function Reference (**Help > Function Reference**).

Did the page load completely during recording?

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Replay failure

If the replay is failing at a particular step, check the step description. VuGen sometimes reads a single space as a double space. Make sure that there are no incorrect double spaces in the string.

Miscellaneous Issues

Out of memory error in JavaScript

Increase the JavaScript memory in the run-time settings.

Increase the JavaScript Memory Size

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate the **Memory Management JavaScript Runtime Memory Size (Kb)** and **Memory Management JavaScript Stack Memory Size (Kb)** options.
5. Increase the memory sizes to 512Kb or higher.

VuGen displays JavaScript errors

If VuGen displays JavaScript errors in the Output pane, enable IE (Internet Explorer) script errors in order to verify that the Javascript itself does not contain errors.

Show Script Errors

1. Open Internet Explorer.
2. Select **Tools > Internet Options** and click the **Advanced** tab.
3. Under **Browsing**, select the **Display a notification about every script error** check box.
4. Rerun the application in IE. If IE displays script errors, then there is a problem with the JavaScript application. If it is not possible to fix the application, you can safely ignore the corresponding replay errors.

   Problems following parameterization

   If you encounter problems only after you have parameterized values, verify that the values are valid for your application. Perform business process with the value of the parameter and verify that the application accepts it.

   Problems with applications that utilize styling actions

   If the client-side scripts of the application use a lot of styling activities, you should record the script again after enabling the **Record rendering-related property values** option. This enables the recording of additional DOM objects.

Enable the "Record rendering-related property values" Option

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Recording > Recording Options** and select the **GUI Properties > Advanced** node.
3. Select the **Record rendering-related property values** check box.

Re-record the Vuser script.

Ajax TruClient Protocol

Ajax TruClient Overview

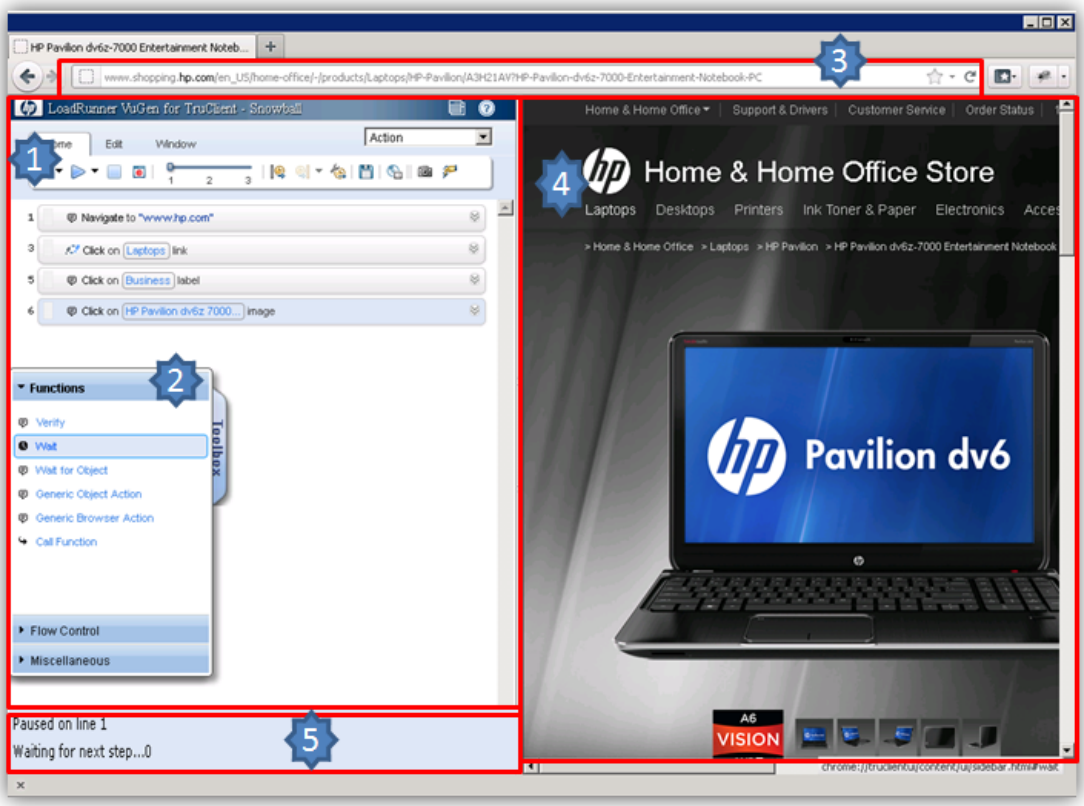
Ajax TruClient Protocol Overview

The Ajax TruClient protocol interactively records scripts as you navigate through your business process. This enables TruClient to easily record and replay dynamic, complex web-based applications and create user friendly scripts. Scripts are created in real-time and steps can be seen in the **TruClient Sidebar** as they are performed. Currently, you can select to record scripts with either the Mozilla Firefox or Internet Explorer browser.

Note: This section documents features available in both the Mozilla and Internet Explorer browsers.

Features that are supported only by a particular browser are indicated by he browser's icon.

The Ajax TruClient User Interface



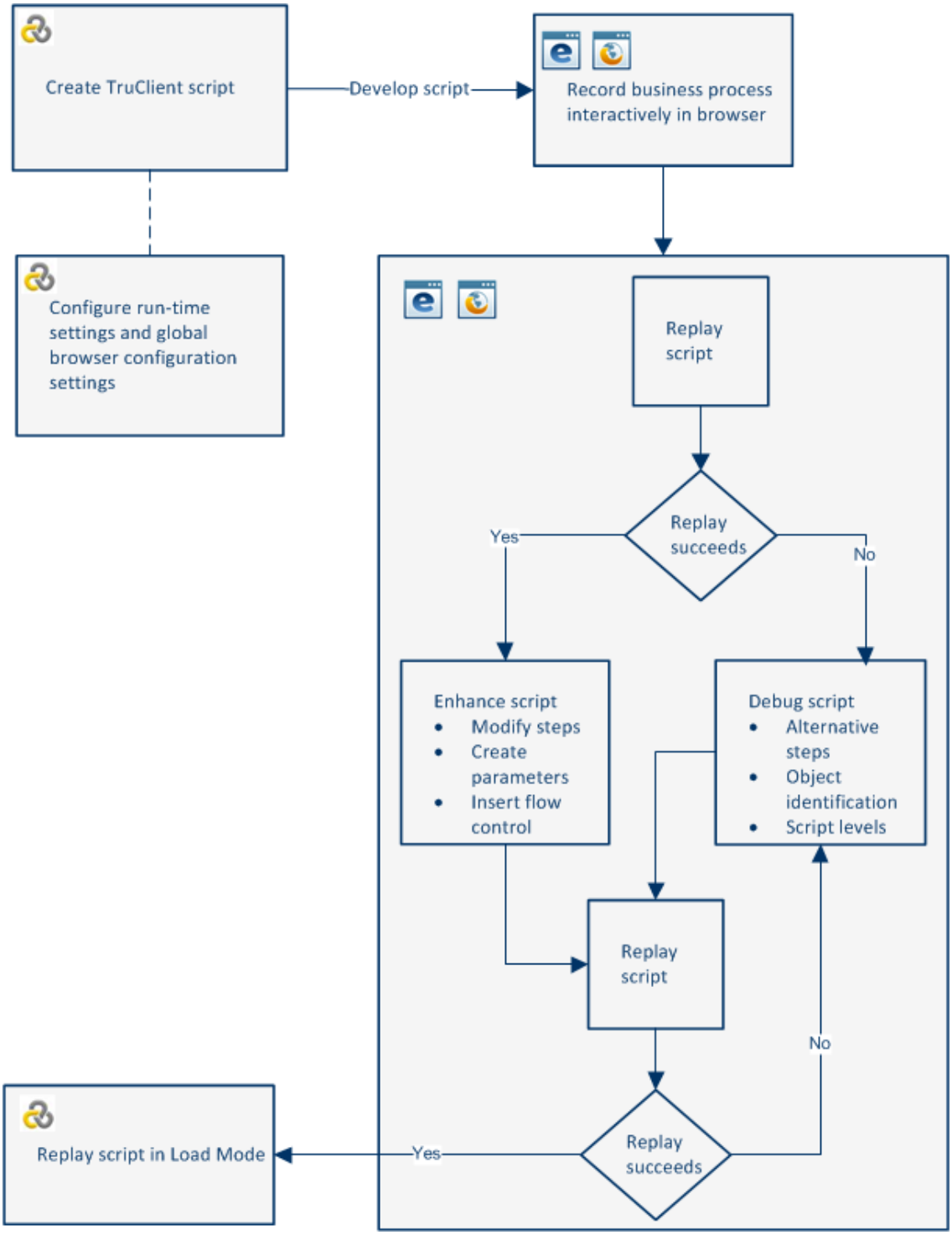
The Ajax TruClient user interface is made up of the following sections:

1. **TruClient Sidebar.** The heart of the interface, contains all the tools you need to develop your Ajax TruClient scripts. For details, see "Ajax TruClient - Developing a TruClient Script" on page 408
2. **TruClient Toolbox.** To enhance your TruClient script by adding steps. The toolbox opens and closes by clicking on the tab, and moves by dragging it up or down. For details see,"Ajax TruClient - Enhancing a TruClient Script" on page 434.
3. **Browser Navigation Bar.** Enter the URL of the application for which you are developing a script.
4. **Application Browser Window** Develop and replay your script interactively.
5. **TruClient Sidebar Status Pane.** Displays status details about the active action in the **TruClient Sidebar**.

You can watch a video demonstration from the VuGen start page.

The Ajax TruClient Workflow

The Ajax TruClient protocol workflow is different from the workflow for other VuGen protocols. The following flowchart illustrates the basic AjaxTruClient workflow:



The following lists some of the main differences between the Ajax TruClient protocol and other VuGen protocols:

- The script is visible in VuGen's script view but it is read only. The script is created and modified in the **TruClient sidebar** in the browser (FF or IE).
- Ajax TruClient scripts are asynchronous. Steps do not have to wait for previous steps to complete. Each step defines an **End Event** which defines the point at which subsequent steps are allowed to start running.
- Ajax TruClient scripts are recorded on the user level, therefore there are no correlations however the main challenge becomes object identification.
- Ajax TruClient scripts are replayed at the user level, therefore each Vuser requires a browser instance with loaded DOM and Javascript. This makes resource footprint larger than the Web (transport based) protocol and also makes it dependent on the DOM and Javascript of the application. For details see, "How to Calculate the Number of Load Generators Required for TruClient Scripts" on page 412.
- All recorded events are saved in the script. Events deemed to be irrelevant are assigned to different script levels and are not replayed unless the level is manually changed by the user.
- Ajax TruClient transactions are defined by step events, not the steps themselves as in other protocols. For example, a step's End Event may allow the script to continue, while a transaction that ends on that step may continue until the step event that defines the transaction is reached.
- The Run-Logic in Ajax TruClient scripts is controlled differently. There is only one action.
- Ajax TruClient step arguments accept JavaScript code as values.

Most of the tasks involved in recording, replaying, and modifying scripts are done using the Ajax TruClient Sidebar for Internet Explorer or Mozilla Firefox .

General Browser Settings (Ajax TruClient)

Each Ajax TruClient script is opened in a browser with a different profile. The browser profile saves user data such as cookies, client certificates, history, cache, and so on. To make changes that are saved in the script profile, make the changes in the browser when interactively developing the script using TruClient. These changes will apply to the current script only.

Browser options that affect interactive mode can be modified in VuGen's **TruClient General Settings** dialog box. All of the settings in this dialog box are imported to each new script as it is created, but affect interactive mode only. When the script is saved, these settings are copied to the run-time settings as well.

Scripts that are run in load mode use the settings defined in the **Load > Browser Settings** node of **Run-time Settings** dialog box. For details, see " Load > Browser Settings Node (Ajax TruClient)" on page 352.

The Firefox extensions settings are imported to each script every time a script is opened in Firefox. For details, see " Ajax TruClient General Settings Dialog Box" on page 405.

Some standard browser options are disabled. These include, but are not limited to, some File, Bookmark, Tools, and Help options. To save bookmarks for use in your scripts, use the " Ajax TruClient General Settings Dialog Box" on page 405 described on " Ajax TruClient General Settings Dialog Box" on page 405.

Private Browsing (Ajax TruClient)

Private Browsing is a browser mode which allows a user to browse without saving information about their session. Some examples of items which are not saved are passwords, cookies, and history.

To more accurately emulate real users,TruClient replays scripts in private browsing mode. This ensures that the browser does not use saved session information when running a script more than one time.

AjaxTruClient Browser for IE

The Ajax TruClient Browser for IE simulates the functionality of an IE browser. This enables you to record and replay scripts as if you were working in an IE environment.

> **Note:** The Ajax TruClient for IE protocol is designed to work with applications running in IE9 in standard mode only.
>
> The TruClient engine depends on specific IE9 browser features that are not supported by Internet Explorer prior to version 9 and are also not supported in IE9 if backward compatibility mechanisms are in use.

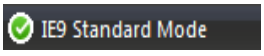
For details on document and browser modes, see Browser modes.

Firebug Lite

Firebug Lite is a third-party utility that provides many valuable development tools. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page. You can access this utility by selecting **Advanced > FireBug Lite** or by pressing **F12** while in TruClient's IE9 browswer.

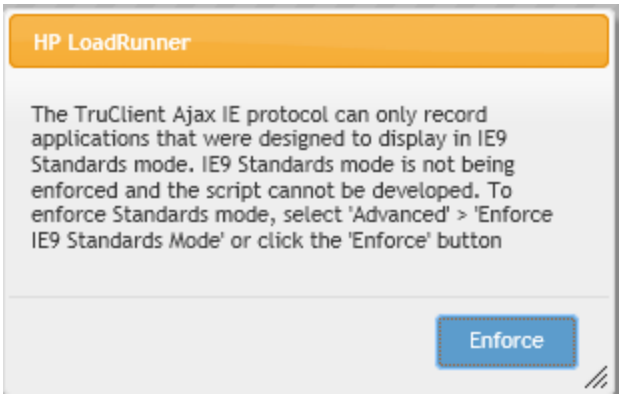
For details on FireBug Lite's features, see What is Firebug?

Enforce IE9 Standard Mode

Ajax TruClient for IE can only record and replay application in IE9 standard mode. TruClient enforces this mode on all pages that you navigate to while you are recording and replaying. You can see the 

icon at the bottom right corner of the browser.

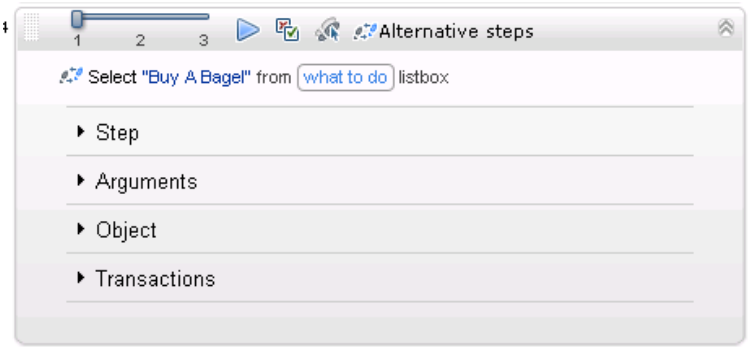
If you wish to explore how pages are rendered in a non-standard IE mode, you can temporarily deactivate enforcement by deselecting **Advanced > Enforce IE9 Standard Mode**. When deselected the following message appears in the Sidebar:



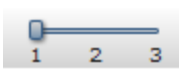
While in the mode you will not be able to develop or edit scripts in the Ajax TruClient IE browser. Click the **Enforce** button in the message box or select **Advanced > Enforce IE9 Standard Mode**, to continue recording and replaying in the Ajax TruClient for IE browser.

Ajax TruClient Step Structure

Ajax TruClient steps are comprised of a number of sections. The sections and elements within each section vary depending on the type of step.



User interface elements are described below:

UI Element	Description
	Drag Step. Enables you to arrange the order of your script by dragging the step to a different location.
	**Expand Step.** Displays the individual components of the steps which include step, argument and object.
	**Script levels selector.** Allows you to view and modify the script level of the step. For more information, see " Ajax TruClient Script Levels" on page 420.
	**Replay.** Replays this step only.
	**Disable/Enable Step.** Steps that are disabled are not replayed. This feature allows you to temporarily remove steps from the script without deleting them.

, continued

	**Optional Step.** Marking a step as optional means that in the event that the step can not find its object, the script continues without returning an error.
	**Alternative Steps.** This icon indicates a step which can be redefined in alternative ways. To redefine the step, click the icon, select the desired step definition, and click Back. For more information, see " Ajax TruClient Alternative Steps" on page 419.

, continued

Step	• **Action.** The action that defines the step. The list of relevant actions is determined by the object roles. • **Object Timeout.** If the object does not appear before this time in seconds, the step returns an error. • **Step Timeout.** If the End Event is not reached by this time in seconds, the step returns an error. The way the script behaves when such an error occurs can be configured in the Run-Time settings dialog box. • **Minimum Time.** The least time in seconds that the execution of the step will take. The value of this field can be either 0, "as recorded" or another manually set number. The step execution will end immediately after the step's end event if minimum time is = 0. A minimum time value greater than 0 forces Ajax TruClient to wait the additional time (if not elapsed already) from the step's end event before moving on to the next step. Ajax TruClient records and stores the time that elapsed between recorded actions and allows you to set the minimum time to "as recorded". • **End Event.** Ajax TruClient defines when a End Event occurs during replay. An **End Event** can be one of the following: ▪ **Action Completed.** Step ends when its action is completed. An example of an action is a button click. ▪ **DOM load.** Step ends when the process of loading a document completes. ▪ **DOM content loaded.** Step ends when the HTML parsing of the document completes. ▪ **Step network completed.** Step ends when all HTTP requests have completed including requests initiated by XMLHttpRequest. ▪ **Step synchronous network completed.** Step ends when all HTTP requests have completed excluding requests initiated by XMLHttpRequest. ▪ **Dialog opened.** Step ends when a dialog is opened. ▪ **Automatic: Not Set Yet.** The automatic end event has not yet been determined. For details, see "How to Synchronize Ajax TruClient Scripts Steps" on page 410
Arguments	Contains step arguments. These arguments differ for different step actions and roles. For a list of the step arguments, see " Ajax TruClient Step Arguments" on page 455.

, continued

Object	• **Roles.** The functions that Ajax TruClient understands about an object. This information is read-only and is updated dynamically depending on how the object is used during recording. The list of available step actions is defined by these roles. • **Name.** A logical name for the object. This does not affect replay and can be modified to enhance readability. • **ID Method.** The method of identifying the object. ▪ **Automatic.** Ajax TruClient's default object identification method. If this method does not successfully find the object during replay, click the Improve Object Identification button, reselect the correct object from the application, and replay the script again. ▪ **XPath.** Identifies the object based on its XPath expression that defines the object in the DOM tree. When you select this option, the next edit box in the display is labeled **XPath** and enables you to select an XPath to define the object. See below for details. ▪ **JavaScript.** JavaScript code that returns an object. When you select this option, the next edit box in the display is labeled JavaScript and enables you to define a JavaScript to define the object. See below for details. • **Related Objects.** Tool to enable Ajax TruClient to identify a target object in relation to an anchor object. For details, see " How to Resolve Object Identification Issues" on page 424.
Transactions	Allows you to create, modify, and view transactions. For more information, see " How to Enhance Ajax TruClient Scripts" on page 437.
XPath/JavaScript	
XPath	VuGen generates from zero to a few suggested XPaths, depending on the object. Click the drop-down arrow next to the XPath edit box to select a suggested XPath for the object. You can manually modify the suggested XPath. To revert to one of the original expressions generated by VuGen, select one of the options from the drop-down again. You can also click the **Regenerate expression** button and select an object. VuGen generates a new set of suggestions based on the selected object.
JavaScript	If Ajax TruClient can generate a suggested XPath for the object, that XPath is entered as the argument in an **evalXPath** function in the JavaScipt field. The **evalXPath** function returns an array of the objects defined by the XPath in the argument. You can modify the suggested XPath in the argument to return a different list of objects, or you can enter a different JavaScript. For example: document.getElementById("SearchButton") returns an element that has a DOM ID attribute of "SearchButton". Ajax TruClient also includes a **random** function that returns a random item from the array that is provided as its argument. For example: `random(document.getElementsByTagName("a"))` **Note:** The evalXPath and random functions are available as object identification methods only. They are not recognized in an Evaluate JavaScript code step.

Ajax TruClient General Settings Dialog Box

This dialog box enables you to set many options that affect the browser as you develop your scripts in interactive mode.

To access	**Ajax TruClient General Settings** button 
Important information	The Ajax TruClient General Settings dialog box can be accessed from the VuGen window or the Ajax TruClient sidebar in interactive mode.
Relevant tasks	" How to Develop Ajax TruClient Scripts" on page 408

Browsing Settings Tab

This tab enables you to configure settings for the Ajax TruClient browser for scripts that you run in interactive mode.

The settings available in the Browser Settings tab are the same settings that are available in VuGen's **Run Time Settings > Load > Browser Settings** node. For a description of the available options, " Load > Browser Settings Node (Ajax TruClient)" on page 352.

Settings that you modify in the Browsing Settings tab affect interactive mode only and apply to all Ajax TruClient scripts. When you save your script in interactive mode, any settings that you modified in the Browsing Settings tab are applied to the Load Mode Browser Settings.

 **Firefox Options**

Interactive Options Tab

User interface elements are described below:

UI Element	Description
Action on Error	**Abort Script**. Abort script on error. **Continue to the next iteration.** Stop iteration on error and continue to next iteration.
Snapshots generation	**Recording snapshots generation** • **Never.** Never generate snapshots during recording. • **Always.** Always generate snapshots during recording. ▪ **Maximum Level.** Enables you to select the script level to generate snapshots (1-3). **Replay snapshots generation** • **Never.** Never generate snapshots during replay. • **On error.** Generate snapshots during replay on error. • **Always.** Always generate snapshots during replay.

Object Identification Assistant	**Enable.** If selected, enables the object identification assistant. The object identification assistant facilitates resolving identification errors during replay. For details, see " How to Resolve Object Identification Issues" on page 424.
Server Parameter	**Replace server with parameter.** If selected, server domain name in navigation steps will be automatically replaced by a parameter.
Console Settings	**Display log messages.** Display console's log messages during replay.
Debug Settings	**Disable Step Into Functions.** Disable display steps of functions during interactive replay.

Encryption Tab

For configuration details, see Mozilla Support for Encryption .

Extension Tab

For configuration details, see Mozilla Support for Extensions.

Bookmarks Tab

For configuration details, see Mozilla Support for Bookmarks.

Internet Explorer Options

Interactive Options Tab

User interface elements are described below:

UI Element	Description
Action on Error	**Abort Script** Abort script on error. **Continue to the next iteration**. Stop iteration on error and continue to next iteration.
Snapshots generation	**Recording snapshots generation** • **Never.** Never generate snapshots during recording. • **Always.** Always generate snapshots during recording. ▪ **Maximum Level.** Enables you to select the script level to generate snapshots (1-3). **Replay snapshots generation** • **Never.** Never generate snapshots during replay. • **On error.** Generate snapshots during replay on error. • **Always.** Always generate snapshots during replay.

Object Identification Assistant	**Enable.** If selected, enables the object identification assistant. The object identification assistant facilitates resolving identification errors during replay. For details, see " How to Resolve Object Identification Issues" on page 424.
Server Parameter	**Replace server with parameter.** If selected, server domain name in navigation steps will be automatically replaced by a parameter.

Client Certificate Tab

Enables you to manually export the client certificate to the script folder .

Ajax TruClient - Troubleshooting and Limitations (General)

This section describes troubleshooting and limitations for Ajax TruClient scripts.

Troubleshooting Communication between VuGen and Ajax TruClient

Error message: VuGen cannot open the Ajax TruClient browser because the port range is busy or invalid.

Recommendation: In order to open a server, a free port is needed. The server is open on the http://127.0.0.1:<free_port>/ url.

The free port can be configured in the following xml files :

%APPDATA%\Hewlett-Packard\LoadRunner\Data\Settings\VuGenProperties.xml

<VuGenProperties>

<BrowserCommunicationServerEndPort value="8090" />

<BrowserCommunicationServerStartPort value="8080" />

If there is a problem with the ports (all are busy or port range is not valid, the start port is bigger than the end port) you can change the range in the configuration file.

If you are unable to resolve this issue contact HP software support.

Error Message: VuGen cannot open the Ajax TruClient browser.This error has occurred because you do not have the proper permissions to establish communication between VuGen and the Ajax TruClient browser.

Recommendation: Ask your administrator or a power user to run this command:

netsh http add urlacl url=http://127.0.0.1:<port_given_in_the_error_message>/ user=<DOMAIN>\<USER_NAME>

Example: netsh http add urlacl url=http://127.0.0.1:8080/ user=DOMAIN\john

Word Verification Function in the Business Process

Many web sites use special Word Verification fields that display some text that the user must enter in order to validate that an actual user is filling out the form.

This is designed to block crawlers, spiders and so on from using the site and taking up valuable system resources.

These fields are especially designed to block automatic tools such as LoadRunner.

In order to complete your business process automatically you must cancel this function on the web site against which you are running the load.

Protocol limitations

Flash, Silverlight, ActiveX limitation for TruClient IE

User actions on UI elements that are based on Flash, Silverlight, ActiveX in general and Java applets technologies are currently not recorded and replayed.

Java applets and Silverlight limitation for TruClient FF

User actions on UI elements that are based on Java applets and Silverlight technologies will are currently not recorded and replayed

Ajax TruClient - Developing a TruClient Script

How to Develop Ajax TruClient Scripts

This task describes the basic steps involved in interactively developing an Ajax TruClient script.

1. Create a Ajax TruClient script from the VuGen toolbar.

   For information about creating a VuGen script see, "How to Create or Open a Vuser Script" on page 114.

2. **Configure the General Browser Settings**

   The Browser Configuration settings allow you to configure settings that apply to all Ajax TruClient scripts. The settings are imported to new scripts as they are created. To edit these settings click the **Ajax TruClient General Settings** button  from the Record toolbar in the VuGen main window and select the **Browser Settings** tab. For details, see " Ajax TruClient General Settings Dialog Box" on page 405.

3. **Configure the Run-Time Settings**

   In VuGen configure the Run-Time settings before recording and performing a load test. To open the Run-Time settings dialog box, click **F4** or select the **Run Time Settings** node from VuGen's **Solution Explorer**. For more information, see "Run-Time Settings" on page 332.

4. ## Start developing the script

   Click 

   to open an interactive recording session in the **TruClient Sidebar** for either the Mozilla Firefox browser or the Ajax TruClient browser for IE.

5. ## Record interactively

   Navigate to the desired starting website and click  **Record** button . All of your actions will be recorded and displayed in the **TruClient Sidebar** on the left as you perform your business process. You can stop recording by selecting the **Stop** button  . You can continue recording from any point in the script.

   To record into different section of the script, right-click a step and select **Record > Record after** or **Record > Record before** to begin recording new steps into the desired location in the script. If you are recording into a group step, select **Record > Record into**. For more information on group steps, see" TruClient Home Tab" on page 413.

6. ## Replay the script

   It is strongly recommended that you replay the script at least two times, correcting any errors that occur during the process. After two successful consecutive replays, you can move on to the next step. If you continue to experience errors, see " How to Debug Ajax TruClient Scripts" on page 422.

   During interactive replay, Ajax TruClient will animate each step progress according to its real running progress by filling the step bar. The step running progress can be for example: finding the test object, performing the action, or synchronizing the step on its end-event. For detail on synchronizing script steps, see "How to Synchronize Ajax TruClient Scripts Steps" on next page.

   In addition, script action details can be seen the **TruClient Sidebar Status Pane.**

7. ## Enhance the script

   You can enhance your script in a number of ways such as inserting parameters, transaction, loops, and verification steps. For task details, see " How to Enhance Ajax TruClient Scripts" on page 437.

8. ## Stop developing

   Click the Save button  to save the script. Close the browser window. Restore the VuGen window.

9. ## Replay the script in Load Mode

   Before you run the script in a Controller scenario, run the script in VuGen's load mode. From the VuGen main window, click the **Replay** button  to replay the script in Load Mode . Progress can be monitored in the Replay log. The browser does not open, and snapshots are not displayed.

   For details on Load Generation, see "How to Calculate the Number of Load Generators Required for TruClient Scripts" on page 412.

10. ## General Tips Regarding Successful Interactive Replay

    Do not resize the browser between record and replay and during replay. This can cause objects

to move and interfere with TruClient's ability to locate them.

Do not switch between applications during interactive replay. Keep the browser in focus.

Note: This is especially important when the Related Objects feature is used, as resizing may change the relative position of the objects.

Note: Any customizations (such as bookmarks) that you make within this instance of Firefox will not be saved globally. This is because VuGen opens each script in a unique Firefox profile. If you want to use Firefox for any use other than creating this script (e.g. browsing the internet), we recommend that you open an additional Firefox window.

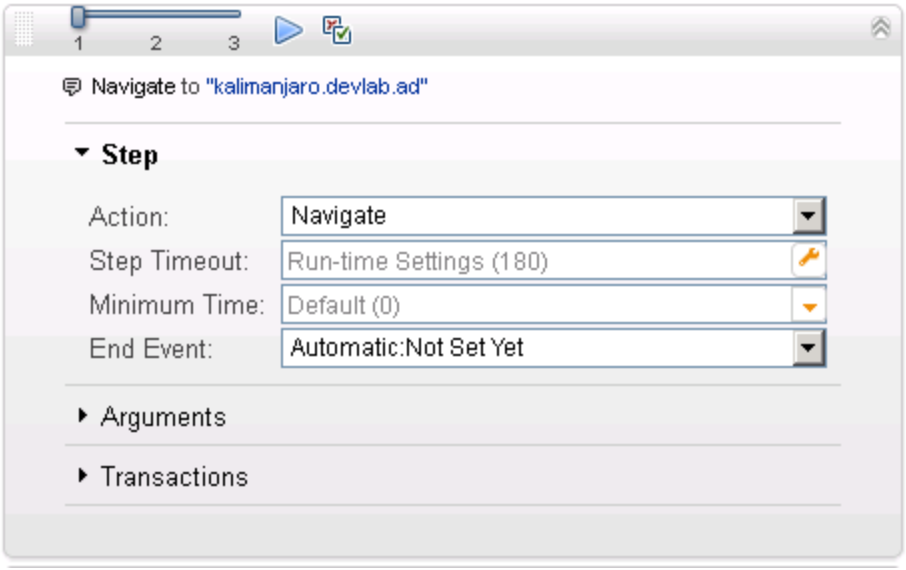
How to Synchronize Ajax TruClient Scripts Steps

This task describes the process of synchronizing steps in Ajax TruClient Scripts.

Note: This task is part of a higher-level task. For details, see " How to Develop Ajax TruClient Scripts" on page 408

Interactively record the business process

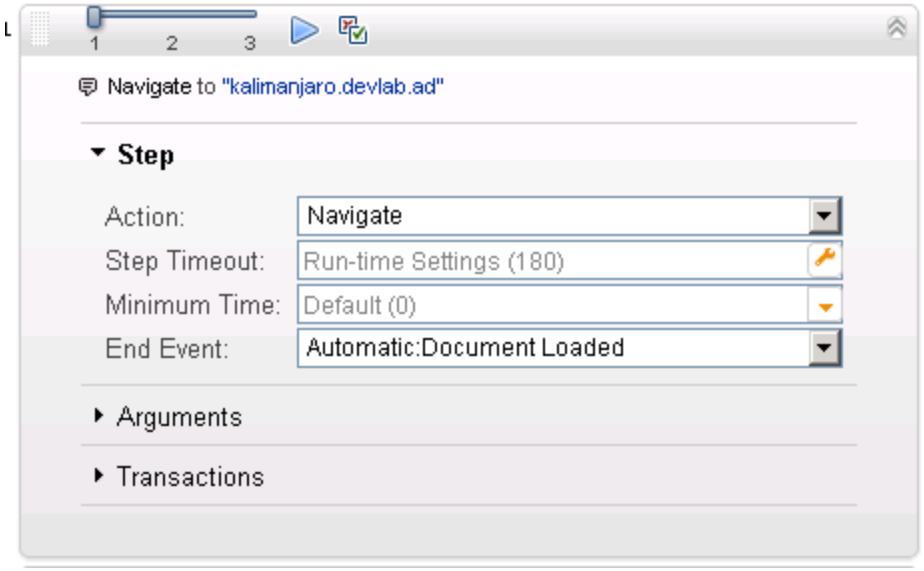
Ajax TruClient scripts are asynchronous which means steps do not have to wait for previous steps to complete. Each step has an **End Event** which defines the point at which subsequent steps are allowed to start running. After the interactive recording, each **End Event** is set to **Automatic: Not Set Yet**.



For details on developing a script in Ajax TruClient, see Step 5 of " How to Develop Ajax TruClient Scripts" on page 408

Identifying the End Event

Before enhancing an Ajax TruClient script, replay the script to synchronize the steps.During the first script replay, TruClient will try to automatically identify the **End Event** for each step.



An **End Event** can be one of the following:

- **Action Completed.** Step ends when its action is completed. An example of an action is a button click.
- **DOM load.** Step ends when the process of loading a document completes.
- **DOM content loaded.** Step ends when the HTML parsing of the document completes.
- **Step network completed.** Step ends when all HTTP requests have completed including requests initiated by XMLHttpRequest.
- **Step synchronous network completed.**Step ends when all HTTP requests have completed excluding requests initiated by XMLHttpRequest.
- **Dialog opened.** Step ends when a dialog is opened.
- **Automatic: Not Set Yet.** The automatic end event has not yet been determined.

If during the first replay, a step initiates the Object Indentification Assistant to resolve object identification, the previous step's **End Event** will most likely be misidentified and TruClient will automatically reset it to **Automatic: Not Set Yet**

Confirming the End Event

During the second script replay, TruClient confirm the Automatic End Event and will assigned any step's End Event that had been reset to **Automatic: Not Set Yet**.

If TruClient is unable to assign an automatic **End Event** during replay, usually due to a network timeout, a warning message will inform you that the **End Event** has been reset to **Automatic: Not Set Yet**. Replay the script to automatically assign the **End Event** or assign the **End Event** manually.

You may need to replay the script several times until all steps have been accurately synchronized.

How to Calculate the Number of Load Generators Required for TruClient Scripts

This task describes how to calculate the number of load generators required to run multiple TruClient Vusers in the Controller.

Note: This task is part of a higher-level task. For details, see " How to Develop Ajax TruClient Scripts" on page 408.

TruClient technology provides you with the ability to quickly and easily record complex business processes. However, because TruClient records at the user level and requires a browser for replay, the more complex an application's client logic is, the more CPU and memory is required to run a Vuser.

Note: The TruClient footprint can be significantly larger than the footprints of other Vuser protocols. This larger footprint will require more CPU and memory capacity than would be required to run a similar business process recorded in another protocol.

Use the following method to determine the required number of load generators:

1. Record a script using Ajax TruClient. For details, see step 5 of " How to Develop Ajax TruClient Scripts" on page 408.
2. Replay a single Vuser in Controller and check the average CPU and the peak memory consumption of the mdrv.exe process by adding a counter for **% Processor Time** and **Private Bytes.** For details on working with Windows Resource Monitors, see Add Windows Resources Measurements Dialog Box.
3. Based on your load generator hardware and the CPU and memory consumption of a single Vuser, calculate the number of Vusers per machine.

   For example:

   Let us assume that each of our load generators has 8 core processors and 8GB of memory.

   Let us also assume that a single Vuser consumes 80MB of peak memory and 10% CPU on average for the specific business process.

   From a CPU perspective, if we limit the CPU consumption up to 70% utilization, we can have 7 Vusers per core processor (70% /10%). If our load generator has a total of 8 cores processors, 8 * 7 Vusers per processor equals 56 Vusers per load generator.

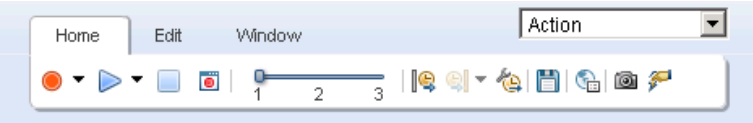
   From a memory perspective, the load generator machine has 8GB memory of which 7GB is available for the Vusers so approximately 87 Vusers per load generator machine (7GB / 80MB).

   Therefore, to meet both the CPU and memory capacity limits, we use the lower number of Vusers and we calculate that for this business process, we can run approximately 56 Vusers per load generator.

TruClient Home Tab

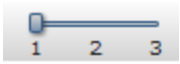
This tab enables you to control the basic flow of the recording process for TruClient scripts.

Actions



To access	Select **Action** from the drop down list on the TruClient sidebar.
Relevant tasks	" Ajax TruClient Step Structure" on page 401

User interface elements are described below:

UI Element	Description
	**Record.** Starts recording the script. Additionally, you can use the arrow to specify whether to record before or after the selected step.
	**Play.** Replays the script. Additionally, you can use the arrow to specify whether to play the selected step only, or to run the script step by step. Running the script step by step pauses the replay after each step. For more information, see " How to Debug Ajax TruClient Scripts" on page 422.
	**Stop.** Stops recording or replaying the script.
	**Toggle Breakpoint.** Toggles breakpoints on the selected step.
	**Script Level.** Modifies the script levels that are visible and replayed in the script. For more information, see " Ajax TruClient Script Levels" on page 420.
	**Start/End Transaction.** Inserts a starting or ending point for a transaction.
	**Transaction Editor.** Opens the Transaction Editor, allowing you to define new transactions and modify existing ones.
	**Save.** Saves the script.
	**General Settings.** Opens the Ajax TruClient General Settings dialog box. For details, see " Ajax TruClient General Settings Dialog Box" on page 405.

	**Snapshot View.** Displays the Snapshot view in the right pane. Adds the following elements to the right pane: • **Snapshot Type.** These buttons enable you to view snapshots taken during different modes: ▪ **Recording.** Display snapshots that were taken for a specific step during recording. ▪ **Interactive Replay.** Display snapshots that were taken for a specific step during interactive replay. ▪ **Load Mode Replay.** Display snapshots that were taken for a specific step during load mode replay ▪ **Play Slide Show.** Display snapshots as a slide show. ▪ **Stop Slide Show.** • **View.** You can select from the following views: • **Single.** Displays the snapshot for a single step. • **Compare.** Splits the screen so you can compare snapshots from different modes. Use the Snapshot View buttons in each pane to select which snapshots to view. Click the **Synchronize Scrolling** button  to synchronize scrolling between the panes. The snapshot error icon  indicates that the snapshot is not current for the step. • **Thumbnails.** Displays the snapshots in thumbnails view. • **Previous/Next.** Navigates to the screenshot for the previous or next step. The corresponding step is highlighted in the script.
	Opens the Event Handler Editor box. For details, see " TruClient Events Handler Editor Dialog Box" on page 444

Functions

To access	Select **Functions** from the drop down list on the **TruClient Sidebar**
Relevant tasks	" How To Create and Use Function and Libraries" on page 442

User interface elements are described below:

UI Element	Description
	Enables you to select the active library.
	**New Library**. Enables you to create a new library.
	**Import Library.** Import a function library from an xml file.

	**Export Library.** Export a function library to an xml file.
	**Delete Library.** Delete a function library.
	**New Function**. Enables you to create a new function. For details , see " How To Create and Use Function and Libraries" on page 442

Context Menu

To access	Select a step and right click to display the context menu.
Relevant tasks	" Ajax TruClient Step Structure" on page 401

UI Element	Description
<Replay Actions>	**Play This Step.** Replay the selected step only. **Play From This Step.** Replay the script from the selected step.
<Record>	• **Before step.** Insert the next set of recorded steps before the selected step. • **Into Step.** Insert the next set of recorded steps into the selected group. • **After Step.** Insert the next set of recorded steps after the selected step.
Toggle Breakpoint	Insert/Remove a toggle breakpoint.
<Transaction Steps>	• **Start Transaction.** Insert a Start Transaction step into the script. • **Surround With Transaction.** Insert a Start Transaction and End Transaction steps around a single or multiple steps into the scripts. • **End Transaction.** Insert an End Transaction step into your script.
<Group Actions>	**Group Steps** Multiple steps are grouped together as a single step. **Group Into** Multiple steps are group into: • **If Clause.** A logical structure that controls the flow of your script**.** • **For Loop Clause** . A logical structure that repeats the steps contained in the loop a specified number of times. • **New Function.** A group of steps, such as a login, that you define as a function
<Edit Actions>	• **Cut.** Cut select step from the script. • **Copy.** Copy selected step in the script. • **Paste.** Paste selected step into the script.
Delete	Delete a step from the script.

Disable/Enable	Toggle between disabling or enabling a step during replay.
Edit Step	Expand the step to display step, argument and transaction properties.
Show	**Step Numbers.** Display step numbers. **Animations.** Enable/disable step animation in the TruClient sidebar.

Ajax TruClient Edit Tab

This tab enables you to cut, copy, and paste steps and data in Ajax TruClient scripts.

To access	Select the Edit tab from the **TruClient sidebar**.

User interface elements are described below:

UI Element	Description
	Cut the selected data or step.
	Copy the selected step or data.
	Pastes before the selected step.
	Pastes after the selected step.
	Pastes into the selected step.
	Deletes the selected step
	Opens the Find dialog box, allowing you to search the script for steps by step name or number.
	Go to the specified step.

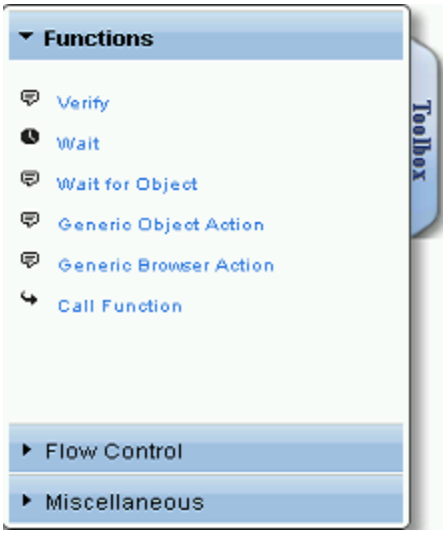
	The following actions can be undone: • Cut • Copy • Paste • Delete • Group • Drag & drop (step) • Add step from the toolbox
	The following actions can be redone: • Cut • Copy • Paste • Delete • Group • Drag & drop (step) • Add step from the toolbox

 Window Tab

This tab enables you to control multiple browser windows for the same script. Select the window that contains the application that you want to bring to focus. This is needed for the debugging phase of script development, for example, when attempting to highlight a step object.

Ajax TruClient Toolbox

The toolbox enables you to add steps to Ajax TruClient scripts. The toolbox opens and closes by clicking on the tab and moves by dragging it up or down.



User interface elements are described below:

UI Element	Description
Functions	• **Verify**. Verify that an object exists in the application. • **Wait.** Wait for a specified number of seconds before continuing with the next step. • **Wait for Object.** Wait for an object to load before continuing with the next step. • **Generic Object/Browser Action**. Blank steps that can be inserted and manually configured. • **Call Function.** Insert a custom function in the script. For details, see " How To Create and Use Function and Libraries" on page 442.

Flow Control	• **For Loop**. A logical structure that repeats the steps contained in the loop a specified number of times. • **If Block.** A logical structure that runs the steps contained in the block if the condition is met. • **Add else.** Click the **Add else** link to add an **else** section to your **If** block. If the condition is not met, the steps included in the **else** section run. • **Remove else.** Removes the **else** section from the **If** block. **Note:** If the **else** section contains steps and you click **Remove else**, the steps are deleted. Copy and paste them into the main body of your script to save them. • **If exists.** A combination of "If Block" and "Verify", a logical structure that runs the steps contained in the block if the condition on a property of the selected object is met. • **If verify.** A logical structure that runs the steps contained in the block if the selected object exists in the application. • **Break.** Causes the loop to end immediately without completing the current or remaining iterations. • **Continue.**. Causes the current loop iteration to end immediately. The script continues with the next iteration. • **Catch Error.** Catches an error in the step immediately preceding and runs the contents of the catch error step. For more information, see " How to Enhance Ajax TruClient Scripts" on page 437. • **Exit.** Exits the iteration or the entire script depending on the specified setting.
Miscellaneous	• **Evaluate JavaScript**. Runs the JavaScript code contained in the step. • **Evaluate JS on Object.** Runs the JavaScript code contained in the step after the specified object is loaded in the application. • **Evaluate C**. Runs the C code contained in the step. • **Comment.** A blank step which allows you to write comments in your script.

Ajax TruClient - Debugging a TruClient Script

Ajax TruClient Alternative Steps

Alternative steps allow you to select from multiple ways to perform the same action in a step. You can modify such steps to perform the given action to debug or enhance your script. For example, in a drop-down list, Ajax TruClient gives you the option of specifying your selection by name or by the number in which it appears in the list.

Steps that have alternative options are labeled with an alternative step symbol . Click it to view the alternative options for that step. Click the desired alternative and select **Back**.

Below is a snapshot of a step in which the second item in a listbox named "Desktop" was selected. The alternative steps feature gives you the option of defining the step based on clicking the link "Desktop", selecting the object "Desktop" from the listbox, or selecting the second item in the listbox.

Ajax TruClient Script Levels

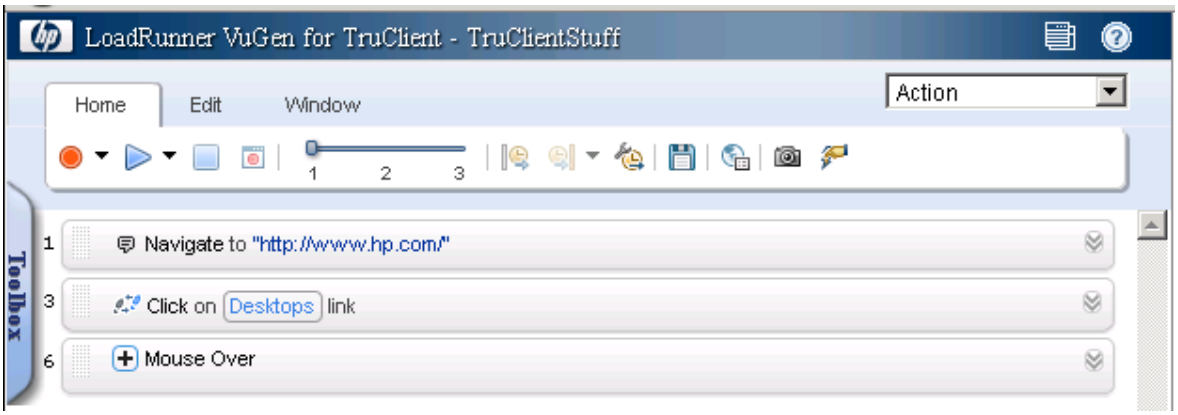
As part of the process of recording a business process, some steps that are performed by the user while recording are not needed during replay. TruClient removes steps it deems to be unnecessary and places them in different script levels. For example, a click step that occurs in an area of the application that has no effect is placed in level 2. TruClient assumes that this step is not significant and will not help the user to emulate a business process on the application. The default view displays level 1 steps only. To view steps from levels 2 and 3 as well, use the slide bar in the home tab. During the replay phase, only steps that are visible are run.

In certain cases, you may want to override TruClient's decisions and manually change the level of a given step. This can happen in cases such as mouse over steps that are needed during replay. TruClient generally views mouse over steps as unnecessary for replay and assigns them to level 3. For more information, see " How to Debug Ajax TruClient Scripts" on page 422.

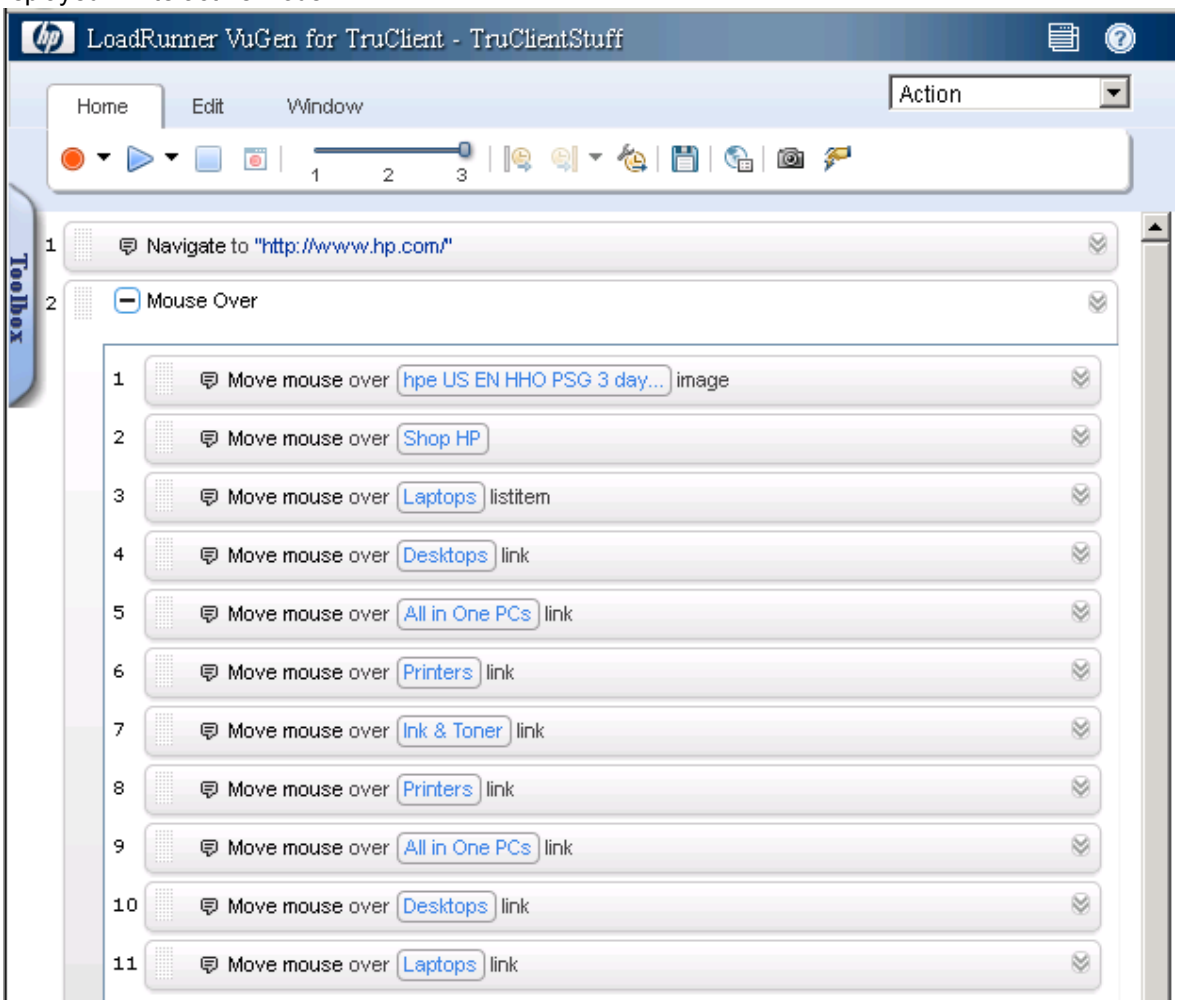
Note: Automatic leveling during replay

The level of a step is normally set during recording according to the importance of the events in the business process. It can happen that an important step will look unimportant and will be placed in a lower script level. This may cause the replay to fail, generating an “object not found” error. During replay,TruClient will check if there are steps in a lower level that can affect the outcome of the current step. If found, the meaningful step will be moved to higher script level.

The following screen shot displays a small script. Note that the step numbers skip from 1 to 3. Step 2 is hidden in a different level.



After changing the display settings by using the slide bar, all steps are now displayed and will run if replayed in interactive mode.



Ajax TruClient Snapshots

Vuser scripts based on the Ajax TruClient protocol utilize VuGen's Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

Ajax TruClient generates snapshots during recording according to the snapshot generation settings. These snapshots can be viewed by hovering the mouse over each step's icon. The snapshots are taken before the step's action is implemented and they are saved as .png files. Click each snapshot to display it in a new browser tab. Make sure that the correct tab is active before replay. Recording snapshots are stored in the snapshot folder.

You can configure snapshot generation during recording and replay by clicking the **General Settings** button  and selecting the **Interactive Options** tab. For details, see " Ajax TruClient General Settings Dialog Box" on page 405.

Snapshots can be viewed in snapshot viewer in VuGen. For details, see "How to Work with Snapshots" on page 57. You can also view snapshots in the Ajax TruClient snapshot viewer. For details, see " TruClient Home Tab" on page 413.

Ajax TruClient can also generate snapshots during load mode according to your specifications in the run-time settings. For details, see "General > Replay Node" on page 355.

Replay snapshots are stored in the results folder and are organized according to the type of replay (interactive or load), the script section, and the iteration.

How to Debug Ajax TruClient Scripts

This task describes different options to debug an Ajax TruClient Script.

View Replay Errors in the TruClient sidebar

If any steps failed during replay, they are marked with an error icon. Hover the mouse over these icons to view descriptions of the errors.

Run The Script Step by Step

You can run your script step by step to view the replay more slowly and in a controlled manner. To run the script step by step, select the down arrow from the replay button in the browser and select **Replay step by step**. Repeat this procedure after each step to continue the step by step replay.

View the Replay Logs

In the VuGen Output Pane, you can view details of your script's replay. Select **Output Pane > Replay** or the **Output Pane > Browser Replay** logs . For details, see the "Output Pane" on page 89.

Insert Toggle Breakpoints

Breakpoints instruct the script to stop running during a replay when in interactive mode. They can be used to help debug your script. To insert a toggle breakpoint, select the desired step and click the Breakpoints  button.

Debug Scripts Using Snapshots

You can use the snapshots generated during replay to debug scripts by viewing the snapshots of the failed step(s).

1. Select the  button from the **TruClient Sidebar** and select the **Interactive Options** tab. Set the **Replay Snapshots Generation** to **On Error**.
2. Replay the script from the **TruClient Sidebar**.
3. Look in the **Output Pane > Replay** or the **Output Pane > Browser Replay** logs for errors. Note the step numbers of the steps that had errors.
4. To view the snapshots from the **TruClient Sidebar**, select a step with an error, and select the  button.

   or

   To view the snapshots from VuGen, select **View > Snapshot Pane**.

You now have a group of snapshots in which errors occurred in the script.

Modify and view script levels

Sometimes, steps that were recorded and are necessary for replay are placed in levels 2 and 3. In this case, you need to manually modify the level of those steps to level 1.

- To modify a the script's replay level, drag the slider in the toolbar to the desired level. Dragging the slider to level 3 displays and replays the steps on levels 1, 2, and 3.
- To move a step to a different level, open the step and click on the step section. Move the slider to the desired level. If the step is part of a group step, both the group step and the individual step must be modified.

For more information, see " Ajax TruClient Script Levels" on page 420.

Insert Wait steps

Sometimes a script will fail to replay because an object in a step is not available when the previous step has finished. You can resolve this by inserting **Wait Steps** into your script which pause the script replay before continuing to the next step. There are two different types of **Wait Steps**:

- The **Wait** step pauses the script for a defined amount of time before continuing to the next step.
- The **Wait for Object** step pauses the script until a specified object is loaded before continuing to the next step.

Note: Wait Steps differ from **Think Time** steps in other protocols. **Think time** controls the time that a VuGen waits between actions. **Wait Steps** pause a script replay until either a specified time elapses or an specified object is loaded.

Wait Steps begin after the **End Event** of the previous step. This means that the previous step may continue to run after the **Wait Step** has been reached.

To insert a **Wait Step**, select **Toolbox > Functions** and drag the **Wait** or **Wait for Object** icon to the desired location in your script. If you add a **Wait** step, configure the interval in the argument

section of the step. If you add a **Wait for Object** step, select the **Click to choose an object** button to select the target object in the application.

Additional Script Debugging Tips

Alert Function

Since all the Ajax TruClient arguments support JavaScript, you use the **Alert** function to display information during script development. You can also reference any DOM element using regular functions, such as location.

Firebug for IE or Dom Inspector for FF

To further improve debugging capabilities, you can install plug-ins such as DOM Inspector and Firebug that can provide additional information on the application object properties.

For more information on Firebug Lite, see" AjaxTruClient Browser for IE" on page 400

Resolving Step Timeouts

Steps may timeout due to several reasons:

- Application is responding slowly, possibly under load. This is actually an important test result.
- Step Timeout is incorrect and should be modified via the Step section of the step properties.
- The end event of the step is incorrect and the step is waiting for an event that doesn't occur. The end event should be changed via the Step section of the step properties.

How to Resolve Object Identification Issues

Object identification presents one of the biggest challenges with recording and replaying Web 2.0 Web applications because objects which have been recorded can move or change content. When recorded objects change dynamically during replay, Ajax TruClient can lose the ability to automatically locate the object.

Ajax TruClient includes very sophisticated mechanisms to overcome this challenge including the Highlight, Improve Object Identification, Replace Object and Related Object, and Object Identification Assistant options. The following steps describe the ways to use these mechanisms.

Note: When identifying objects for applications that have been recorded in multiple windows, make sure that the correct window is selected in the **TruClient Sidebar > Window Tab > Replay Window**.

Highlight, Improve Identification, Replace, Related Objects all require the user to select an object in the application. There are cases in which various actions are required in the application to make the object visible such as mouse over and mouse click. In these cases use the CTRL+ALT+F4 option to suspend the TruClient object selection mode until you've brought the object into view and press CTRL+ALT+F4 again to select the object.

Tip: After you perform any of the changes, replay the single failed step in question and only afterwards replay the whole script again. This will help verify whether the change has solved

the issue you encountered.

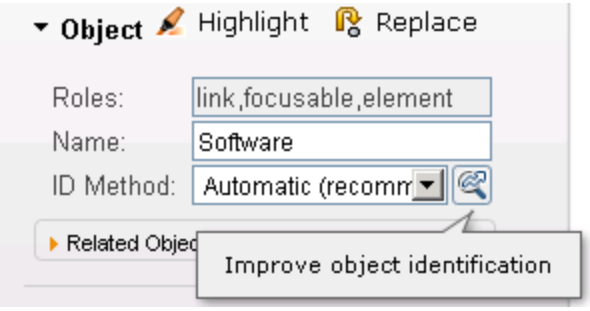
The following steps describe the ways to resolve object identification issues.

Highlighting an object

Regardless of which method of object identification is used, you can use the highlight button, located in the Object section of the step, to check if an object is visible in the application at any time. If the object is not found this may be an issue of pacing and timing. If the object cannot be found, an error message is displayed.

Improve Object Identification

If the Highlight option fails, use the Improve Object Identification.



This option is located in the Object section of the step, next to the ID Method drop down. This will let Ajax TruClient relearn the properties of the object and compare them to the properties learned during recording. Based on the differences, the necessary adjustments can be made. Depending on how dynamic the application is, you may need to use the Improve Object Identification function more than once.

Once you have done this, try replaying the step again to verify the problem has been solved.

Alternative Steps

Alternative steps allow you to view instances in which there are multiple ways to perform the same action in a step. If Improve Object Identification fails, try using one of the alternative steps.

For example, you may be clicking on an option in a drop down list in which the text changes based on some value.

If you try to click based on the text, the step may fail.

If you use an alternative step that selects the item in the list based on the ordinal value of the option within the list, the click will succeed regardless of the text.

Note: Before selecting one of the alternatives, try highlighting the object used by the alternative step and replaying it. This way you'll make sure the alternative step is replaying the necessary action.

Modify the Object Identification Method

You can modify the way Ajax TruClient identifies the object by modifying the object identification method in the Object section of the step properties. The following options are available:

- **Automatic.** Ajax TruClient's default object identification method. The Automatic method allows Ajax TruClient to use its internal advanced algorithms to locate the object. If this method does not successfully find the object during replay, click the Improve Object Identification button and replay the script again.
- **XPath.** If Automatic identification fails, even after using Improve Identification or Related Objects (described below), try using the XPath identification method. This method identifies the object based on an XPath expression that defines the object in the DOM tree. Click the drop-down arrow next to the **XPath** edit box to select a suggested XPath for the object. You can manually modify the suggested path. To revert to one of the original expressions generated by Ajax TruClient, select one of the options from the drop-down again.

For example, if you need to select the first search result, regardless of the term being searched for, using XPath identification may help.

- **JavaScript.** JavaScript code that returns an object. For example: document.getElementById ("SearchButton") returns an element that has a DOM ID attribute of "SearchButton".

Using the JavaScript identification method you can write JavaScript code that references the returned document and can use CSS selectors and other standard functions.

For example, the page returned by the server contains multiple links with the same "title" attribute (search results) and we want the script to randomly click on one of the available links.

Object identification for this case, using the JavaScript identification method, may look something like this:

```
var my_results = document.querySelectorAll('a[title="SearchResult"]');

random(my_results);
```

Modify the script timing

Sometimes objects may not be found because of timing and synchronization issues. For example, the script may be looking for an object that was in the application, but the script replayed too quickly and already progressed to another page. If you suspect that the object is not being found because of a timing or synchronization issue, you can insert Wait steps. For more information, see " How to Debug Ajax TruClient Scripts" on page 422.

Relating objects to other objects

If the **Improve Object Identification** function does not solve the issue and neither do any of the alternative steps, try using the **Related Objects** option.

If an object becomes difficult to identify on its own, you can label the object based on a different, more stable object. For example, you can select an object which is not dynamic and "relate it" to the target object. Relations are defined visually, relating objects according to their distance in pixels from other objects. Relations are defined per ID method, per object. If more than one relation is defined for an ID method of a given object, both relations must locate the same object for the step to pass. Ajax TruClient then uses this object to help locate the target object. To use this function, expand the step, select **Object > Related Objects**, and click the add button  . Follow the directions to create a relation. Verify that it has worked by highlighting both the object and its related object.

Tips:

- Use this feature only if other identification methods have failed as it may be more resource intensive.
- Use the minimum search area to improve performance.
- Related Objects are sensitive to window sizing. Resizing may alter object positions and relationships. This should be taken into account.
- Each identification method (Automatic, XPath, and JavaScript) has its own set of related objects. These related objects are not shared between identification methods.
- If several relations exist they all need to be found in order for the identification to succeed.

Replacing an object

If you selected the wrong object during recording, or an object has permanently changed you can replace it with a different object without replacing the step. This effectively resets the step, deleting changes made to the original step such as relations. Expand the step, select **Object**, and click the **Replace** button  . Select the new object and replay the script.

Replace Object will tell Ajax TruClient that the object currently referenced in the step is incorrect. Ajax TruClient will remove any current knowledge of the object and learn the object you select from scratch.

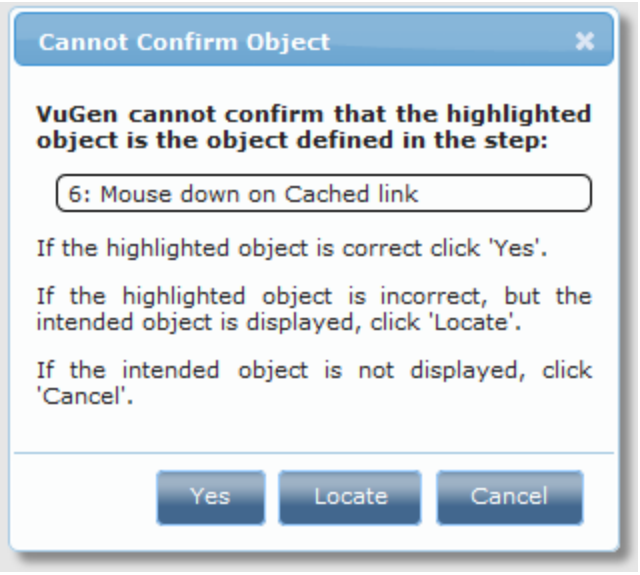
Therefore, you should only use the Replace Object option if the object you used during recording was the wrong one.

Object Identification Assistant

Object identification can fail during replay for several reasons. For each reason, Ajax TruClient will launch the object identification assistant to try to resolve failed identification.

Cannot confirm object

Ajax TruClient suspects a specific object to be the desired object but it cannot be positively identified. The suspected object is highlighted on the screen, and the following assistant dialog box appears:

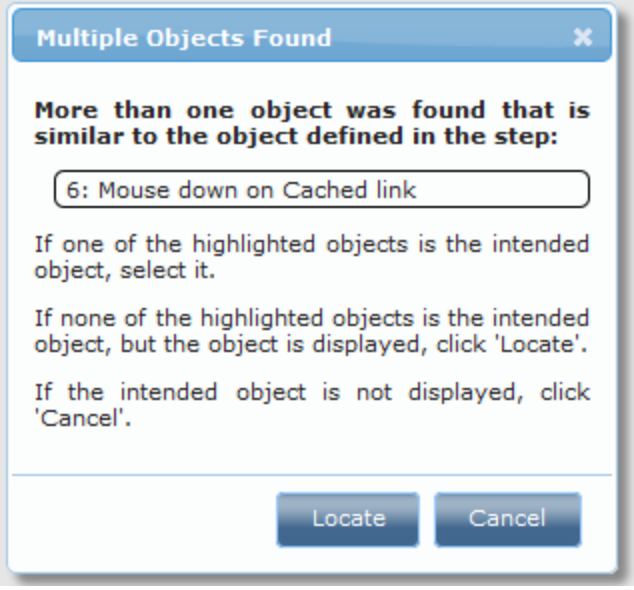


Options:

- **Yes**. The suspected (and highlighted) object is the correct object.
- **Locate**. The suspected object is not the right object. You will need to identify the correct object in the application.
- **Cancel**. Stop the replay.

Multiple objects found

Ajax TruClient found several objects that match the identification of the desired object. All suspected objects will be highlighted on the screen, and the following assistant dialog box will appear:

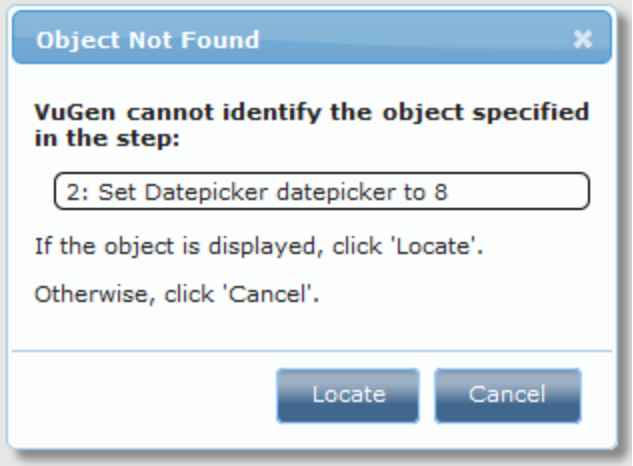


Options:

- The correct object is one of the marked objects. Click the object in the application to specify the correct one.
- **Locate**. The suspected object is not one of the highlighted objects.You will then need to highlight the correct object in the application.
- **Cancel**. Stop the replay.

Object not found

Ajax TruClient could not find the desired object. The following assistant dialog box will appear:



Options:

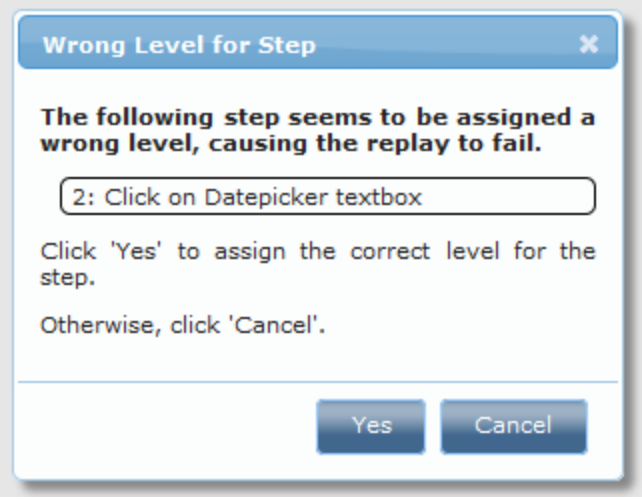
- **Locate**. Locate the object. You will need to highlight the correct object in the application.
- **Cancel**. Stop the replay.

Wrong level on step

Ajax TruClient found that a step in a higher level is needed for the identification of the object of current step. This is common for mouse over steps that are usually recorded in level 3 but might be needed for click steps that are recorded in level 1.

When Ajax TruClient recognizes this dependency during **recording**, the mouse over will move automatically to level 1.

When Ajax TruClient recognizes this dependency during **replay,** the following assistant will appear:



Options:

- **Yes**. Move the needed step to level 1.
- **Cancel**. Stop the replay.

Troubleshooting Object Identification Issues

How Related Objects Can Help the XPath and JavaScript Identification Methods

The XPath and JavaScript identification methods may return multiple elements depending on the expression used.

For example, if the XPath value is //button, and the web page in question includes multiple button elements, multiple objects will be returned.

To return a single object you can add a Related Object that will narrow down the identification.

Interactive Replay Fails with Object Not Found Error

If replay stops with an error that says the object wasn't found try the following:

- Select the failed step and press Highlight. If the correct object isn't highlight there's an object identification issue and you should use Improve Identification to improve object identification.
- If the object is highlighted it may be that step was reached before the object appeared. Add a Wait or Wait for Object step before the problematic step.

Sometimes you may need to choose an Alternative Step that is provided in order to solve the issue. For example, you may be clicking on an option in a drop down list in which the text changes based on some value. If you try to click based on the text, the step may fail.If you use an alternative step that selects the item in the list based on the ordinal value of this option within the list, the click will succeed regardless of the text.

Interactive Replay Fails Due to Object Not Found Although Highlight Locates the Object

If the replay fails even though the Highlight option finds the correct object, this may be a case of pacing.

The object takes a little longer to load and the step is executed faster. Therefore, during execution the step cannot locate the object although, during debugging, the Highlight option on the failed step finds the object.

In this case it is recommended to "slow down" the script so there's enough time for the object to load. Use one of the following options to do this:

- Change the Object Timeout of the failing step. This is available via the Step section on the step's properties.
- Add a Wait or Wait for Object step before the failed step.

Replay Fails to Select an Item from a List

One of the common reasons for this is that the names of the items in the list are dynamic.

For example, the list may include a list of cities based on the text entered so far (auto-complete).

Based on the text types the list constantly changes.

There are two ways to solve this issue:

- Use an alternative step that selects an item from the list using the ordinal identifier instead of the text of the actual item.
- If the text is only partially dynamic you can use a regular expression to locate the required item based on partial text matching.

Troubleshooting Ajax TruClient Scripts

How to Check for Specific Text Including Branching

One option is to add a Verify step from the Functions section of the Toolbox. In this step you can select various validation settings such as the object, the text to look for, and so on.

When you want to perform certain actions based on whether a validation succeeded or failed you add a Catch Error step from the Flow Control section of the Toolbox.

This way you can make sure the step continues even if the validation failed and within the Catch Error group you can define the set of steps that should be executed if the validation fails.

You can also take a more programmatic approach to validations. Using JavaScript you can access the DOM and validate any property you wish to verify. You can then add a conditional Break or Exit step (available via the Flow Control section of the Toolbox) based on this verification.

You can also check for the required text directly from the IF statement. The 'condition' argument of the IF statement is simply JavaScript code. You can use JavaScript code that accesses the window global object of the application under test. This can be done by referring to window.

Then you can manually verify if the text exists within the current page. For example, assuming a single frame application, you can write something like:

```
window.document.body.textContent.indexOf("Off") == -1
```

Where "Off" is the text you're looking for and -1 indicates that the text was not found.

The code in question is application specific.

You can optimize the code if you have further knowledge of the application (by getting the specific element).

How to Check for Specific Text that is Case Insensitive

By default, the Verify step is case sensitive. For example, looking for 'Test' will fail if 'test' is found.

If you'd like the Verify step to be case insensitive do the following:

- In the Verify step set the Condition argument to "Regular Expression"
- To check if the string 'test' is contained in the text regardless to case you can use:

```
RegExp("test", "i")
```

How to Select a Random Option from a List

Set the Ordinal argument to 0. TruClient will automatically select a random option from the list.

For example, let's assume you have an auto-complete list that shows a list of cities based on the typed text. You've currently selected the second option and the step is: Select option #2 from City autocomplete.

All you need to do is open the Arguments section of the step properties and change the Ordinal argument to 0. The step will now be: Select a random option from City autocomplete.

This option is very important when the typed text is a parameter and therefore you have no easy way of knowing in advance what values exist in the list and how many there are.

How to Use External Functions in the Script

Add your JavaScript and C functions to the JS-Function.js and C-functions.c files that are part of the script and appear in the left navigation pane in VuGen.

JavaScript functions can be called directly from the Ajax TruClient script, as all arguments and parameters support JavaScript. You can also add an Evaluate JavaScript or LR.evalC step from the Toolbox for this purpose.

To call C functions add an Evaluate C step from the Toolbox.

Some of the Events and Actions do not Appear in the Recorded Script

Try solving this in one of the following ways:

- Ajax TruClient records all the events in the application. The event you're looking for may be in a different script level from the one being displayed.
  - You can tell that additional steps exist in other script levels if the steps in the viewed level are not numbered consecutively.
  - The current script level is set using the slider in the toolbar:

    Try looking for the missing event or step in the other levels by changing the slider value.
  - Once you have found the required event you can change its level and make it part of level 1.
  - Change the script level back to level 1 and try to replay again.
  - Please consult the Ajax TruClient documentation for a full explanation of the script level concept.
- You can manually add a step to the script.
  - From the toolbox select the Generic Object Action and customize the step to perform the required action.

Dragging of a Slider or a Map does not Replay Correctly

If drag does not work (e.g. set option of slider, drag of map) and the result does not bring the control to the appropriate place try the following:

- Try using one of the Alternative Steps and see if it helps.
- Set the values manually until they meet your needs (e.g. the precise number of pixels you'd like to drag in each of the relevant directions).
- Try using the "Drag to" capability (by changing the Action of the Drag step in the Step section of the step properties). This way you can drag your object to a relative position from another object.

Polling for an Object

Create a loop that includes the following steps:

- A step that performs an action on an object.

- A Catch Error section that includes a Continue step.
- A Break step.

The Catch Error section will make the loop continue until the object is found and the step succeeds.

How do I Create a WHILE Loop

A "For" loop has 3 arguments: Init, Condition, and Increment.

A "While" is basically a "For" loop with only the Condition argument.

To create a while loop add a "For" loop (using the context menu or the toolbox), delete the Init and Increment arguments and specify the Condition.

Troubleshooting Load Mode for TruClient Scripts

How to Make the Script as Stable as Possible

One of the characteristics of Web 2.0 applications is objects with dynamic attributes.

For example the ID of an object may have different values every time the application is accessed.

To make sure the script is stable and can run even when these changes occur, the following steps are recommended:

- After you have successfully replayed the script, make sure you replay it again several times in the Firefox interactive script development sidebar.
- If any of these replays fails, use the various methods for improving object identification (e.g. Improve Identification, Related Objects, and Alternative Steps).
- If the application objects change based on certain events (e.g. IIS Reset), make sure these events occur between replays, to validate the scripts stability.
- Use the Run option in VuGen to run the script again in load mode and validate that it is ready to be incorporated into a load test.

How to Fix a Script that Replays in Interactive Development but Fails During Load

It is highly recommended to use the Run option in VuGen to test the script in load mode before actually using it in a load test. The Run function plays the script the way it would be played in a load test and enables you validate it before including it in a load test.

If the script fails during Run in VuGen or in the actual load test but functions correctly in interactive replay please follow these steps:

- Look at the replay log to find out what the error is.
- If the error is "object not found" this may be due to pacing. Load mode replay is faster than interactive replay and the object may not be loaded in the application quickly enough.
- You can resolve this using one of the following methods:
  - Use the "Inter-step interval" run-time setting which allows the user to specify the time (in ms) between steps. This setting will be used for ALL the steps in the script.

- Add a Wait or Wait for Object step before the step that failed. This will slow down the replay.

- Some applications may be sensitive to the browser size. When the script runs under load, a certain predefined window size is used which may cause object location failures.

  To solve this you can either add a Resize step to the script or set the initial width and height for non-interactive mode replay. This can be done using the 'Non-interactive window size' setting, located under 'Other Settings' in the Run-Time Settings.

How to Debug the Script in Load Mode

Consult the log and get a better understanding of the error.

To view the state of the application when the error occurred go to Run-Time Settings and select Snapshot on Error and replay the script. The snapshot will be generated when the error occurs and will be saved under the results folder based on the run logic (**Results > Load > [Action-Name] > [Iteration-Number]**).

You can also add additional messages to the log by using the LR.log function inside the script itself.

For more information on this function please consult the LoadRunner Ajax TruClient documentation.

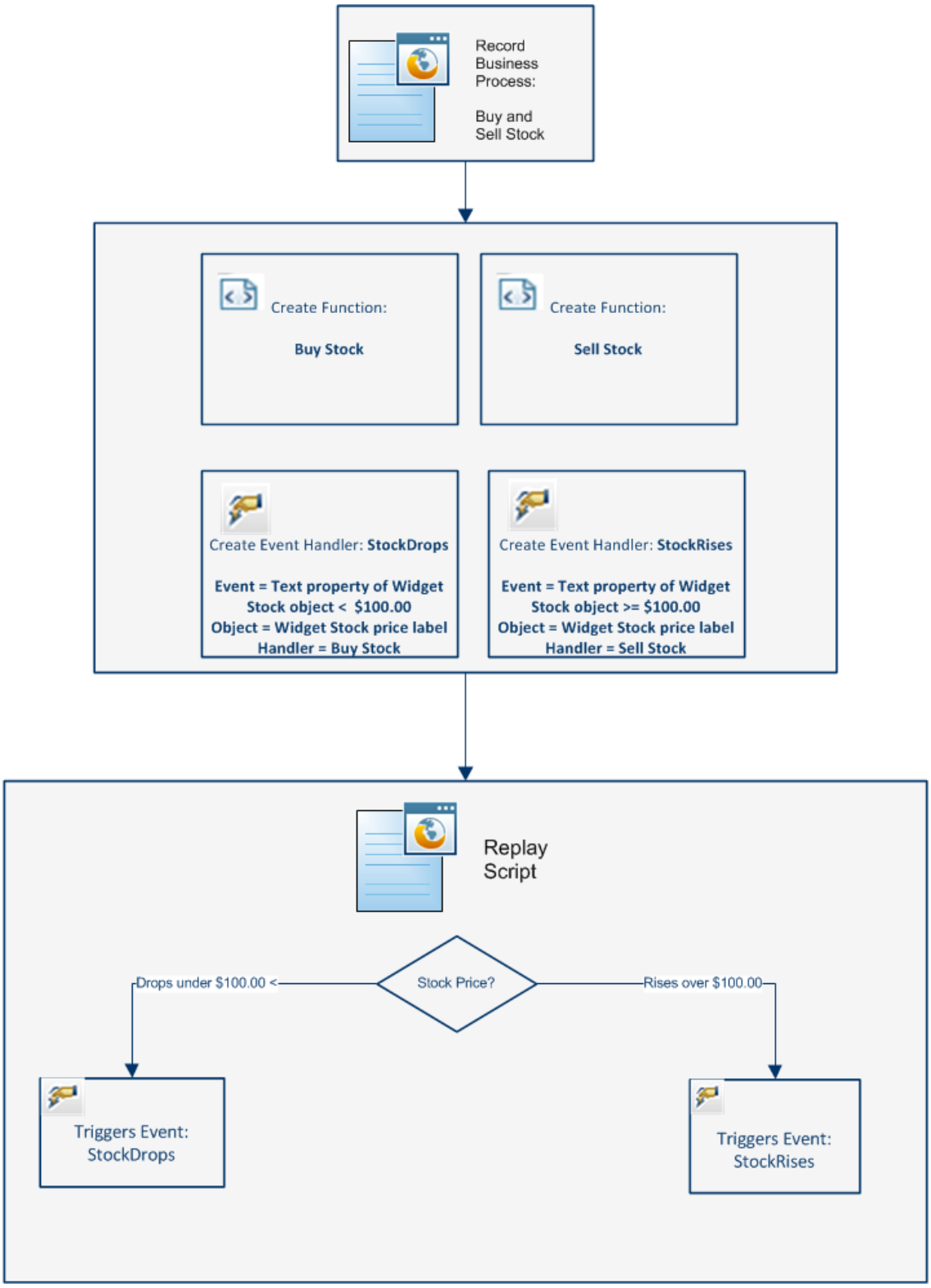
Ajax TruClient - Enhancing a TruClient Script

Ajax TruClient Event Handlers

Event handlers are designed to manage events that can occur at any time during the run of the script. The event trigger is set to an object appearing in the application with or without an additional specified property. The handler (action) of the event is a function selected from the script's function libraries.

During the replay of the script, when moving from one step to the next, Ajax TruClient checks to see if a defined event occurs and if it does, Ajax TruClient runs the function associated with the event.

The flowchart below illustrates the Event Handler workflow using the business process example of buying and selling of stocks.



For details, see :

- " How To Use The Events Handler" on page 439
- " TruClient Events Handler Editor Dialog Box" on page 444

Ajax TruClient Functions and Function Libraries

A TruClient function is a group of steps, such as a login, that you define as a function. Functions are stored in libraries that can be reused multiple times in a script. Each library can contain multiple functions.

Each library can be either local or global. A local library can be accessed by the script that created it. A global library can be accessed by all Ajax TruClient scripts. Additionally, a global library can be saved on the network and shared between many users.

For details on creating and using function and function libraries, see " How To Create and Use Function and Libraries" on page 442.

Working With JavaScript in Ajax TruClient Scripts

The following section contains considerations for recording Ajax TruClient scripts.

JavaScript Support

The arguments listed in the Arguments section of each step are all JavaScript based and can accept JavaScript expressions which will be evaluated during script replay. It is important to remember that to provide a string value, quotation marks are required. For example: City will be interpreted as a variable whereas "City" or 'City' will be evaluated as a string.

All other sections such as Step, Object, Transactions, are not JavaScript based, are not evaluated as JavaScript and do not support JavaScript expressions. The only exception is object identification using JavaScript.

However, object identification variables and step variables do not share the same context. Variables that are defined in one context are not recognized in the other. To use a variable in object identification that is defined in a step, add the prefix **ArgsContext** before the variable name. For example, if the variable **firstname** is defined as a value for an argument of a step, it cannot be used in object identification. To use the **firstname** variable in object identification, refer to the variable as **ArgsContext.firstname**.

How Can I Learn More about JavaScript

JavaScript is the scripting language of the Web. JavaScript is used in millions of Web pages to add functionality, validate forms, detect browsers, and much more. The Internet is full of resources for learning JavaScript and can be located using search engines.

An example of some tutorials and references:

http://www.javascriptkit.com/jsref/

http://www.w3schools.com

http://www.learn-javascript-tutorial.com/

Using Regular Expressions

To use regular expressions, there are two options:

- Use the '/' notation: Replace the quotation marks of a string with a slash.

  For example:

  /LoadRunner/ is a regular expression that will match any string that contains the word "LoadRunner" in it.

- If you need to dynamically create a regular expression (e.g. using a parameter), you can use the regular expression constructor and specify the string. For example, the equivalent of the above example is RegExp("LoadRunner").

The full list of supported regular expressions can be found here:

https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/RegExp

How to Enhance Ajax TruClient Scripts

There are a number of optional enhancements that can be added to scripts beyond the basic workflow. This task describes the enhancements and how to use them.

Modify Steps

Modify step arguments and objects by selecting the desired step and expanding the options. This expands the step and allows you to modify the objects and properties. For a detailed list of the step structure, see " Ajax TruClient Toolbox" on page 417.

Insert Flow Control Steps

Loops repeat selected portions of the script until certain criteria is met or for a specified number of times. To insert a loop, select **Toolbox > Flow Control > For loop**. For more information, see " How to Insert and Modify Loops" on page 441.

Insert If blocks or If-else blocks and exit steps

To conditionalize a portion of the script, you can insert If or If-else blocks. To insert an If block, select **Toolbox > Flow Control > If block**. To add an else condition, click the **Add else** link next to the If step title. For more details, see " Ajax TruClient Step Arguments" on page 455.

Exit steps cause a script to exit the iteration or the entire script. These can be used with If statements to exit a script or iteration when a specified condition occurs. To insert an exit step, select **Toolbox > Flow Control > Exit**.

Insert comments

You can insert comments into your script by selecting **Toolbox > Misc** and dragging the **Comment** icon to the desired location.

Insert Transactions

You can add transactions by using the Transaction Editor. To open the Transaction Editor click the Transaction Editor  button from the Home Tab or click Ctrl + Alt + F7. Ajax TruClient transactions

function differently from other protocols because of the asynchronous nature of Ajax TruClient steps. Transactions are defined based on start and end steps and step events. Due to this definition, a transaction's end can be triggered before the true end of a step.

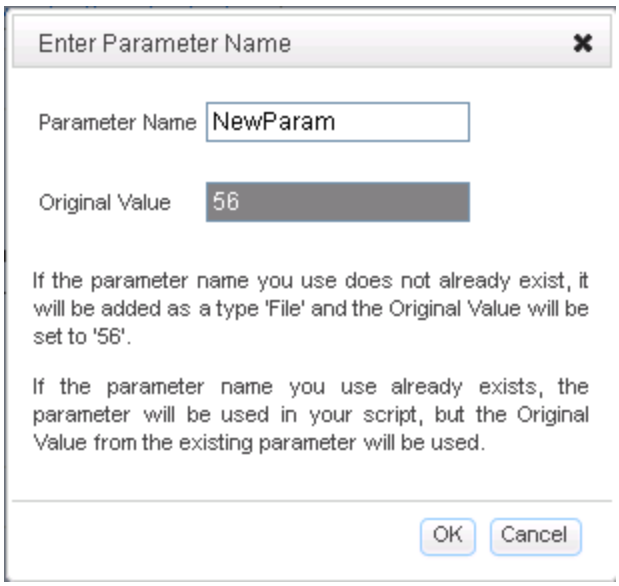
Create Parameters

Parameters for Ajax TruClient scripts can be created in the standard method as for all protocols.

Parameters can be referenced and created within step arguments or Eval JavaScript steps.

To parameterize a value:

1. Select a field in the step Arguments area.
2. Highlight either the entire string or part of the string.
3. Right-click and select **Replace with a Parameter**.
4. In the Enter Parameter Name dialog box, enter the name of a new or existing parameter.



If the parameter does not exist in the **Parameter List**, it is created and the default parameter type is set to **File**. You can go to the **Parameter List** to add or delete values for the specified parameter. By default, the original value is included in the list of values for the new parameter. For details, see "How to Work with Existing Parameters" on page 241.

The following is an example of how to use a parameter in your script. This string can be used as the Argument in an Evaluate JavaScript step:

During replay it returns the current value of the parameter **paramname.**

```
LR.getParam("paramname")
```

During replay it assigns the parameter **paramname** the value **value1**.

```
LR.setParam("paramname", "value1")
```

For example, let's assume we have an auto-complete list that has a different set of values based on the text that's typed.

Trying to select an option in the list based on the text of the option is bound to fail once the typed text is defined by a parameter. The options are changed every time the parameter value is updated.

In these cases the step that uses an ordinal value is more appropriate

For more details on parameters, see the section on "Parameters" on page 227.

Insert Catch Error Steps

Catch error steps are group steps that run their contents if the previous step contains an error. Additionally, the error is "caught" and is not returned. You can define catch error steps to catch any error, or a specific type of error. If there are two catch error steps in a row, they both apply to the same step. To insert a catch error step, select **Toolbox > Flow Control > Catch Error**.

Verify that an objects exist

To verify that a string or object exists in the application, you can insert a verify step:

1. Select **Toolbox > Functions** and drag the **Verify** icon to the desired location.
2. Click the object in the verify step.
3. Select the object you want to verify.

Insert Generic Steps

You can insert a blank step and manually configure it. To insert a generic step, select **Toolbox > Functions > Generic Object/Browser Action**,drag and drop the step to the desired location. Next, expand the step, and enter the desired step properties. Generic Object Actions perform an unspecified action on an object. Generic Browser Actions perform an unspecified action on the browser such as go back, reload and switch tabs.

How To Use The Events Handler

This task describes how to create and use events with the Global Events Handler. For an illustration, see " Ajax TruClient Event Handlers" on page 434.

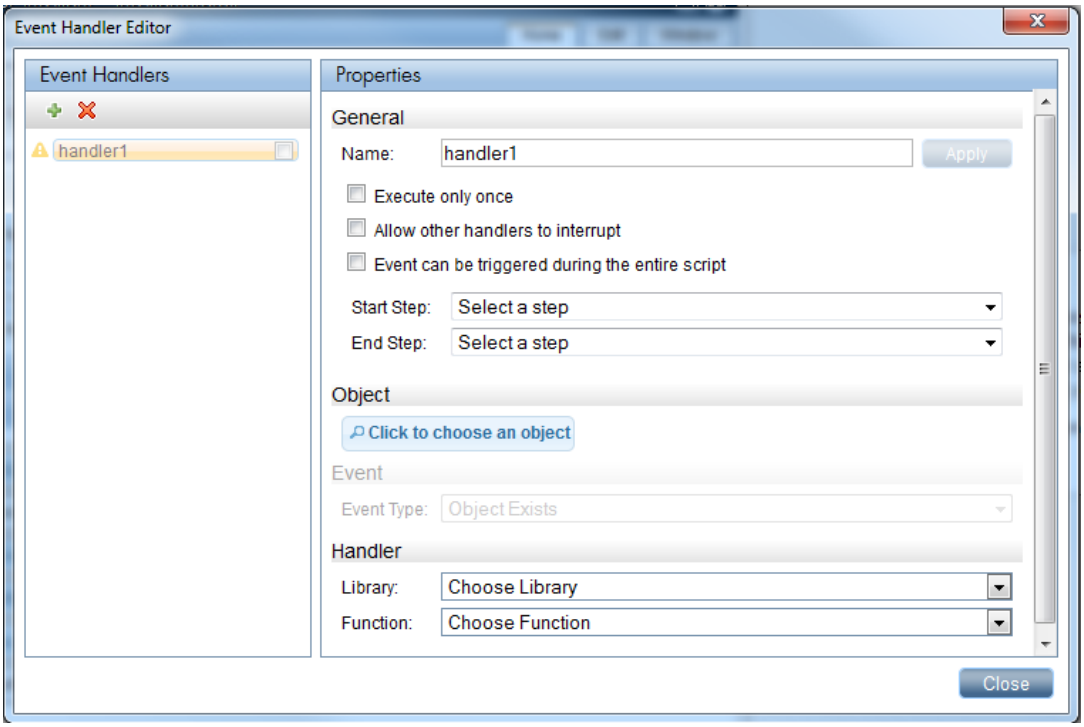
Prereq uisites

The Global Events Handler runs functions if a certain event occurs during the replay of the script. Before you create handlers, you must create the associated functions. For details, see " How To Create and Use Function and Libraries" on page 442.

Create an Event Handler

1. Click the  button from the TruClient Toolbar. This will open the Event Handler Editor

.



2. Click the button from the Event Handler pane.

3. Define the properties of the handler.

   You can specify if you wish to the handler to run at any time during the script or only between certain steps. For details on this and other properties, see " TruClient Events Handler Editor Dialog Box" on page 444.

Define object associated with event

Select the object in the application with the Click to choose an object button on which the event will occur.

For example, select the text box for the Widget stock price if you wish to trigger the function SellStock, if the stock prices rises over $99.99.

Once the object is selected you can select the Highlight button to verify that the correct object was selected. Additionally, you can select the Replace button to replace the selected object with another one. For details, see " How to Resolve Object Identification Issues" on page 424.

Define the event

You can select to run the handler if the object exists or if, object's properties meet specific conditions. For details, see " TruClient Events Handler Editor Dialog Box" on page 444.

Assign handler to the event

Select the function library, the function and specify argument values. For details, see " TruClient Events Handler Editor Dialog Box" on page 444.

Enable the handler

To enable the handler during script replay, select the check box next to the handler in the Event Handler Pane.

How to Insert and Modify Loops

Loops repeat selected portions of the script until certain criteria is met or for a specified number of iterations. You can insert loops and loop modifiers from the **Flow control** section of the **Toolbox**.

For Loops

For loops perform the steps surrounded by the loop until the end condition is met or the code reaches a break statement. Loops arguments use JavaScript syntax. To insert a for loop, select **Toolbox > Flow Control >Functions > For Loop**.

Break statements

Break statements indicate that the current loop should end immediately. For example, if a break statement is encountered in the second of five iteration in a for loop, the loop will end immediately without completing the remaining iterations. To insert a break statement, select **Toolbox > Functions > Flow Control > Break**.

Continue statements

Continue statements indicate that the current loop iteration should end immediately. The loop condition is then checked to see if the entire loop should end as well. For example, if a continue statement is encountered in the second of five iterations in a for loop, the second iteration will end immediately and the third iteration will begin. To insert a continue statement, select **Toolbox > Functions > Flow Control > Continue**.

How to Insert Custom JavaScript and C Code into Ajax TruClient Scripts

This task describes how to insert code into an Ajax TruClient script. You can insert code into a pre-existing step as part of a step argument or insert steps that are completely comprised of external code (C or JavaScript).

1. ### Insert code into a pre-existing step

   You can insert JavaScript code into pre-existing steps in the arguments fields of most steps. This allows you to perform any number of customizations.

2. ### Insert steps composed entirely of code

   You can insert steps comprised entirely of code into your script. To do so, select **Toolbox > Miscellaneous** and drag the **Eval Javascript**, **Eval C,** or **Eval JS on Object** icon to the desired location. The **Eval JS on Object** step runs the JavaScript code after the specified Object has loaded. We recommend avoiding Eval C and using JavaScript instead wherever possible. The user can refer to this object as the variable "object" in the JavaScript code within the step.

   Example:

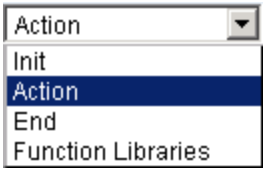
   The following code creates a variable called amount that generates a random number between 1 and 5. You can then use this variable in the argument fields of other steps.

   ```
   var amount=Math.floor(Math.random()*5)+1;
   ```

How To Create and Use Function and Libraries

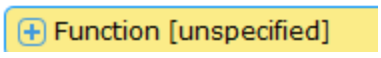
This task describes how to create and use function libraries and functions in TruClient Scripts.

Create a function library and function from the TruClient sidebar

1. Select **Functions** from the 

drop down list in the VuGen TruClient tab.
2. Select an existing library from the 

drop down or click the  button from the Function toolbar to create a new library.
3. Click the  button to create a new function. This will insert a new unspecified function.

4. Click the  button to expand the function.
5. Define the step including function name and end event.

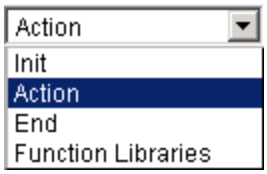
6. Define the function arguments using the **Argument Editor**. Argument names should be meaningful so that when you are using the function it is clear what value you need to specify.
7. Define the transaction using the **Transaction Editor**. For details, " Transaction Editor Dialog Box (Ajax TruClient)" on page 460.

Create a function within a script

1. Highlight the steps in the script to include in the function. To select multiple steps, press CTRL.
2. Right-click on a highlighted step and select **Group Into > Function** which opens the **Create a New Function** dialog box. For details, see " Create New Function Dialog Box" on page 459.
3. To assign steps that contain arguments, expand the argument section, and insert the function `FuncArgs.<argument name>.`.

> **Tip:** To insert javascript as the argument value, click the JS button to access the Javascript editor.

Create a global library

1. Select **Function Libraries** from the 

drop down from the **TruClient Sidebar**.
2. Select an existing library from the 

drop down in the **TruClient Sidebar**.
3. Click the button to export the library to a location on your file directory as an xml file.

> **Note:**
>
> - When working with a global library, you can save changes to the library by clicking the button.
> - If you save the library to a network location, other users can click the button to import the library.
> - Click the button to disconnect the library from global mode. Any changes that you make in local mode are saved within the script.

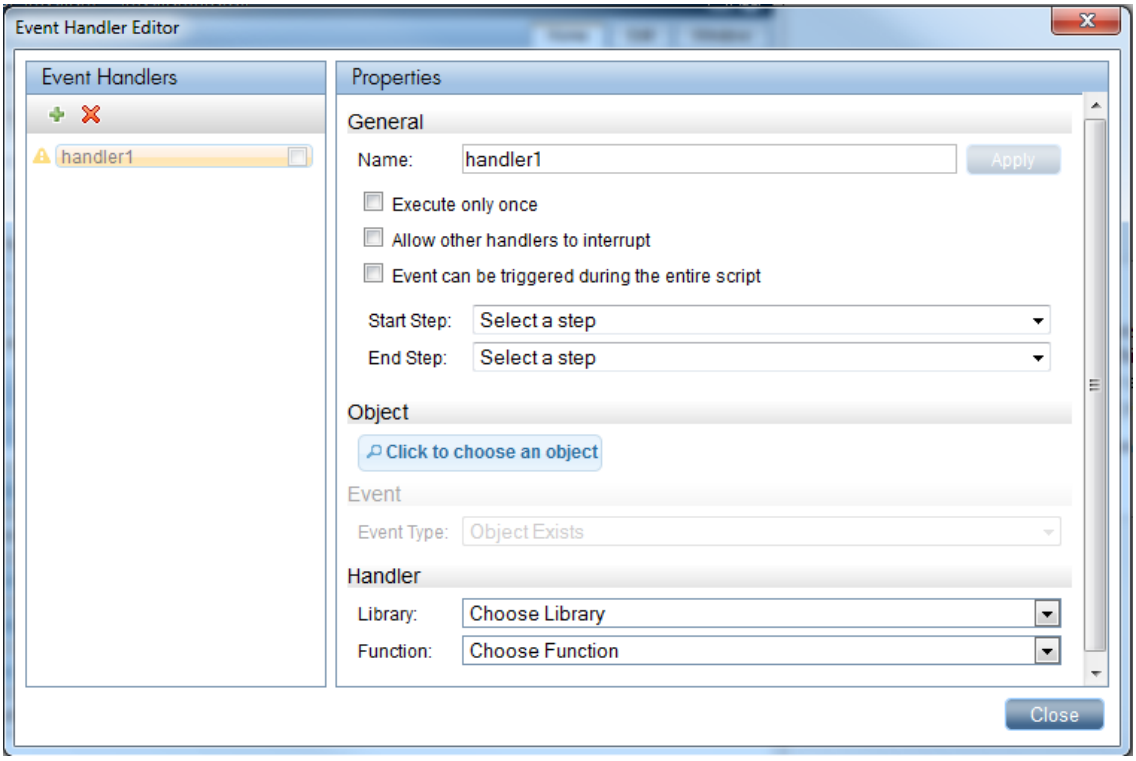
Use a function in the script

1. Click the TruClient **Toolbox** tab.
2. Select the **Functions** tab.
3. Drag and drop **Call Function** to the correct location in your script.
4. Click the **Call Function** link to expand the function.

5. Specify the library and the function in the step section.
6. Specify the arguments' values in the augments section. For details, see "Create a function within a script" on previous page. (Optional)
7. Specify transactions in the transaction section. (Optional) For details, see " Transaction Editor Dialog Box (Ajax TruClient)" on page 460.

TruClient Events Handler Editor Dialog Box

This dialog box enables you to define events handlers and its properties.



To access	Click the button from the TruClient sidebar
Important information	Events handlers are designed to handle events that can occur at any time during script replay.
Relevant tasks	" How To Use The Events Handler" on page 439 " How To Create and Use Function and Libraries" on page 442
See Also	" Ajax TruClient Event Handlers" on page 434

User interface elements are described below:

UI Element	Description
Event Handler Pane	 **Add Event Handler**. Add an event to the event handler.  **Delete Event Handler**. Delete an event from the event handler.

UI Element	Description
Properties	**General** • **Name.** Enables you to name the event. • **Execute only once.** Select check box to execute the event once. • **Allow other handlers to interrupt.** Select check box to enable other events to run during the run of this event. You can select either: • **Event can be triggered during the entire script**. Select check box to enable event to be triggered during entire script. or • **Start Step.** Enables you to select a start step from which point the event handler can be triggered. • **End Step.** Enables you to select a end step after which the event handler cannot be triggered. **Object** • Click to choose an object Enables you to select an object from your application that will trigger the event handler. • **Roles**. Displays the roles which determines what operations can be done on the object. For example, an “element” role has operations like “click” and “mouse over". The “textbox” role will also have a “type” operation. Other roles: element, foucasable, textbox, checkbox, filebox, radiogroup, listbox, multi_listbox, slider, datepicker, video, audio, browser. • **Name.** Enables you to specify the name of the object. • **ID Method** **Automatic (Recommended)**. Enable TruClient to resolve object identification. **XPath**. Enables you to insert an xpath query to identify the object **JavaScript**. Enables you to insert JavaScript code to identify the object. • **Related objects.** Enables you to associate your object with another object in your application that facilitates object identification during replay. **Event** • **Event type** You can select between two event types: **Object Exists**. If object exists during replay, the event handler is triggered. **Property exists on Object.** If the property of an object meets the defined criteria, the event handler is triggered.

UI Element	Description
	Handler • **Library**. Enables you to select the library containing the function. • **Function**. Enables you to select the function. • **Arguments**. Enables you to specify the argument values.

Ajax TruClient Functions

The following functions can be inserted as values in Ajax TruClient step elements.

Method	Description	Arguments	Related Function
The arguments in this row can be used in all methods		• **object.** The step's object as defined in the application. • **window.** Points to the global window object of the application. • **document.** The global document object of the application.	
IO.createDir(path)	Creates the specified folder. If needed, creates all the folders necessary to complete the path.	**path.** The absolute path of the folder.	

Method	Description	Arguments	Related Function
IO.delete(path)	Deletes the specified folder or file. If a folder is specified, deletes all the files in the folder, including sub-directories.	**path.** The absolute path of the folder or file.	
IO.isExist(path)	Returns True if the specified folder or file exists and False if it does not. Returns False for a dis-connected or unau-thenticated mapped drive.	**path.** The absolute path of the folder or file.	
IO.read(path, charset)	Returns all the data from the specified file. Converts the data from the specified charset to unicode.	**path.** The absolute path to the file. **charset.** The charset of the file, if it is not UTF 8.	

Method	Description	Arguments	Related Function
IO.write(path, string, append, charset)	Writes the string to the specified file. If the file does not exist it is created.	**absolute path.** The absolute path to the file. **string.** The string to write to the file. **append.** Boolean. **True.** (default) Append the string to the end of the file. **False.** Overwrite the file with the string. **charset.** The charset of the file, if it is not UTF 8.	
LR.advanceParam (parameter)	Advances the specified parameter to the next value in the file.	**parameter.** The name of the parameter to advance. Must be a parameter of type file or unique number.	lr_advance_param
LR.getUserIP	Returns the IP address when IP spoofing is enabled and the script is running in load mode.		
LR.setParam(name, value)	Saves a string to a parameter, creating the parameter if it does not exist.	**name.** The name of the parameter in which to save the value. **value.** The value.	lr_save_string
LR.crossTransactionStart(name, value)	Begins a transaction in a Vuser script that will be ended in another Vuser script.	**name.**The name of the transaction. **value.**A pointer to a parameter list.	LR.cros-sTransactionEnd

Method	Description	Arguments	Related Function
LR.crossTransactionEnd(name, value,status)	Ends a transaction in a Vuser script that has been started in another Vuser script.	**name.** The name of the transaction. **value.** A pointer to a parameter list. **status.** One of the following:"Pass" or "Fail".	LR.cros-sTransactionStart
LR.transactionDuration(name)	Returns the duration of a specific transaction.	**name.** The name of the transaction	
LR.getParam(name)	Returns the value of the specified parameter.	**name.** The parameter name.	lr_eval_string
LR.getLRAttr(name)	Returns the value of the specified mdrv command parameter.	**name.** The name of the command-line parameter.	lr_get_attrib_*
LR.evalC(funcname)	Runs the specified function whose definition is found in the specified file.	**funcname.** The function name. Use the `#include` command to include the file where the function is defined.	

Method	Description	Arguments	Related Function
LR.log(text, level)	Logs a message.	**text.** The message. **level.** One of the following: • "Error", • "Warning", • "Standard", • "Extended", • "Status". **example:** LR.log ("text", "Error");	lr_debug_message
LR.decrypt(text)	Returns the text after decryption.	**text.** The encrypted text.	lr_decrypt
LR.userDataPoint(name, value)	Records a user-defined data point for analysis.	**name.** The name of the data point. Do not begin a data-point name with any of these strings: HTTP, NON_ HTTP, RETRY, mic_, stream_, mms_ **value.** The numeric value.	lr_user_data_point
evalXPath(XPath)	Returns an array of the objects defined by the XPath in the argument.	**xpath.** The xpath query.	
LR.startTransaction (name)	Starts a LoadRunner transaction.	**name.** The name of the transaction to start.	
LR.endTransaction (name, status)	Ends a LoadRunner transaction.	• **name.** The name of the transaction to end • **status.** One of the following values: “Pass”, “Fail”, “Auto”	

Method	Description	Arguments	Related Function
LR.vuserStatusMessage(string)	Indicates which Vuser is handling a specific script.	**string.** Any string. **Example:** LR.v-user-StatusMessage ("FlightStatus")	
Utils.clearCookies()	Removes all cookies currently stored by the Vuser.		web_cleanup_ cookies
Utils.clearCache()	Clears the contents of the cache simulator.		web_cache_ cleanup
Utils.getEnv(name)	Returns the value of the specified environment variable. Returns an empty string if the specified variable does not exist.	**name.** The name of the environment variable.	
Utils.import(path)	Evaluates the specified JavaScript file in the arguments context.	**path.** The absolute path to the JavaScript file.	

Method	Description	Arguments	Related Function
Utils.isEnvExists(name)	Returns **True** if the specified environment variable exists and **False** if it does not exist.	**name.** The name of the environment variable.	
Utils.setEnv(name, value)	Sets the named environment variable to the specified value. If the variable already has a value it is overwritten.	• **name.** The name of the environment variable. • **value.** The value to set the environment variable to.	
Utils.addAutoFilter(filter, isIncluded	Adding a filter to one of the black list or white list lists of URLs. The URL of each HTTP request will be checked against those lists – first the black list, and if not empty, the white list. HTTP requests that do not pass the check will be blocked.	• **filter.** String representing the URL. • **isInclude.** True value indicates the white list, false otherwise.	

Method	Description	Arguments	Related Function
Utils.removeAutoFilter(filter, isIncluded	Remove a filter from one of the black list or white list lists of URLs.	• **filter.** String representing the URL. • **isInclude.** True value indicates the white list, false otherwise.	
Utils.cleanupAutoFilters ()	Remove all filters from both the black list and white list lists of URLs.		
Utils.addAutoHeader (header, value, merge)	Adding an HTTP header to every consecutive HTTP requests following this function call.	• **header.** The name of the header to be added. • **value.** The value of the header to be added. •  **merge.** True value indicates to merge the value with an existing header by the same name, false indicates to overwrite it.	
Utils.removeAutoHeader (header)	Stops adding an HTTP header to every consecutive HTTP request following this function call.	**header.** The name of the header to be stopped from being added.	

Method	Description	Arguments	Related Function
Utils.cleanupAutoHeaders ()	Removes all HTTP headers and stops adding HTTP headers to every consecutive HTTP request following this function call.		

Ajax TruClient Step Arguments

The following table displays the step arguments categorized by role. Mandatory arguments are marked with a red star to the left of the argument name in the user interface. All arguments can accept JavaScript code and LoadRunner functions as values. For a list of LoadRunner functions, see "Ajax TruClient Functions" on page 447.

Role	Action	Arguments
element	Evaluate JavaScript	Code: JavaScript code
element	Mouse Actions: Mouse Down, Mouse Up, Mouse Over, Click, Double Click	• **Button.** The mouse button that is clicked. • **X Coordinate.** The offset location of the action relative to the upper left corner of the object. This number must be positive. If not specified, the default is the center of the object. • **Y Coordinate.** The offset location of the action relative to the upper left corner of the object. If not specified, the default is the center of the object. • **Ctrl Key.** Whether or not this key is pressed during the action. • **Alt Key.** Whether or not this key is pressed during the action. • **Shift Key.** Whether or not this key is pressed during the action.

Role	Action	Arguments
element	Drag	• **Button.** The mouse button that is clicked. • **X Offset.** The amount of pixels to drag the object on the x axis. A positive number indicates a drag to the right. • **Y Offset.** The amount of pixels to drag the object on the y axis. A positive number indicates a drag down. • **Path.** List of coordinates representing user drag path. Do not modify this argument.
element	Drag To	• **Target Object.** The step object is dragged to this target object. • **X Offset.** The offset from the top left of the target object in the x axis. This number must be positive. • **Y Offset.** The offset from the top left of the target object in the y axis. This number must be positive.
element	Get Property	• **Property.** The property whose value will be stored in the specified variable. The list of properties available depends on all the roles of the object. The following are the default properties available for all objects: ▪ **Visible text.** The visible text of the item, corresponding to the DOM textContent property. ▪ **All text.** The entire text of the item, corresponding to the DOM textContent property. ▪ **Inner HTML.** The inner html markup of the object, corresponding to the DOM innerHTML property. ▪ **Variable.** The name of the variable in which to store the specified property value.

Role	Action	Arguments
element	Verify	• **Value.** The string or number to verify. • **Property.** The object property whose value will be verified. The list of properties available to verify depends on all the roles of the object. The following are the default properties available for verification on all objects: ▪ **Visible text.** Items that are visible in the application. ▪ **All text.** Items that are in the application but are not necessarily visible. Items in this category are contained in DOM property textContent. ▪ **Inner HTML.** Items contained in the DOM property innerHTML. ▪ **Condition.** The relationship between the value and property arguments.
element	Wait for Property	• **Value.** The value of the specified property that the step will wait for, before the step passes. • **Property.** The object property whose value the script will wait for. The list of properties available for which to wait, depends on all the roles of the object. The following are the default properties available for all objects: ▪ **Visible text.** Items that are visible in the application. ▪ **All text.** Items that are in the application but are not necessarily visible. Items in this category are contained in DOM property textContent. ▪ **Inner HTML.** Items contained in the DOM property innerHTML. ▪ **Condition.** The relationship between the value and property arguments.
focusable	Press Key	• **Key name.** Enter or Space.
text box	Type	• **Value.** What is typed. • **Clear.** Clear the text box before typing. The default is true. • **Typing Interval.** The average time in milliseconds between keystrokes.
checkbox	Set	• **Checked.** Set the checkbox to either checked (T) or unchecked (F).

Role	Action	Arguments
listbox	Select	• **Text.** The selected string. • **Ordinal.** The order of the selected item in the list. If the text argument is also specified, than this argument refers to the instance of the specified text value in the listbox. An ordinal of 0 generates a random value.
radiogroup	Select	• **Text.** The selected string. • **Ordinal.** The order of the selected item in the list. If the text argument is also specified, than this argument refers to the instance of the specified text value in the listbox. An ordinal of 0 generates a random value.
filebox	Set	• **Path.** The selected path.
slider	Set	• **Value.** The value that the slider is set to.
datepicker	Set Day	• **Day.** An integer between 1-31 representing the day of the month.
browser	Activate	• **Ordinal.** Defined as an integer. Moves the specified browser window to the foreground.
browser	Activate Tab	• **Ordinal.** Which tab (integer) to activate.
browser	Close Tab	• **Ordinal.** Which tab (integer) to close.
browser	Add Tab	• **Location.** The URL to navigate to in the newly opened tab.
browser	Navigate	• **Location.** The URL to navigate to.
browser	Go Back	• **Count.** The number of pages to go back.
browser	Go Forward	• **Count.** The number of pages to go forward.
browser	Resize	• **Width.** The new width. Leaving this blank means do not resize the width. • **Height.** The new height. Leaving this blank means do not resize the height.
browser	Scroll	• **X Coordinate.** The new x coordinate. Leaving this blank means do not scroll along the x axis. • **Y Coordinate.** The new y coordinate. Leaving this blank means do not scroll along the y axis.
browser	Dialog - Confirm	• **Button.** Ok or Cancel.

Role	Action	Arguments
browser	Dialog Prompt	• **Value.** The string to enter. • **Button.** Ok or Cancel.
browser	Dialog - Authentication	• **Username.** The username to enter. • **Password.** The password to enter. • **Domain.** The domain to enter. • **Button.** Ok or Cancel.
browser	Verify	• **Value.** The value of the property to verify. • **Property.** The property to verify. You can verify the following properties of a browser object: • **Title.** The title of the browser window. • **Location.** The location of the browser window. • **Condition.** The relationship between the value and property arguments.

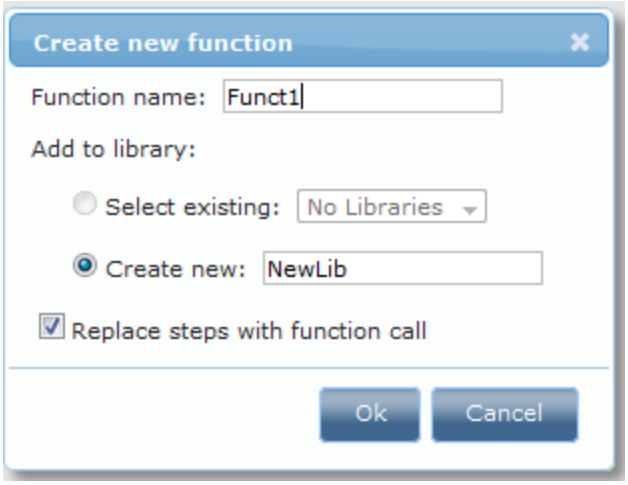
Ajax TruClient Properties

These properties can be used as arguments in Ajax TruClient functions.

Property	Description
LR.userId	The user ID as it appears in the MDRV command line. MDRV is the main process that runs all protocols.
LR.groupName	The group name as it appears in the MDRV command line. MDRV is the main process that runs all protocols. If the process was started by VuGen its value is 0.
LR.scenarioId	The scenario ID as it appears in the MDRV command line. MDRV is the main process that runs all protocols. The scenario ID exists only if MDRV was started by Controller, if it was started by VuGen its value is 0.
LR.outputDir	The user output folder that contains all the output for the script. For VuGen the output folder and script folder are the same. For Controller they are different. The returned path includes the last folder separator.
LR.scriptDir	The user script folder. You can store external files in the script folder such that when you want to include them in your script, you can append the filename of your external file to LR.scriptDir.

Create New Function Dialog Box

This dialog box enables you create a new function and assign the function to a library.



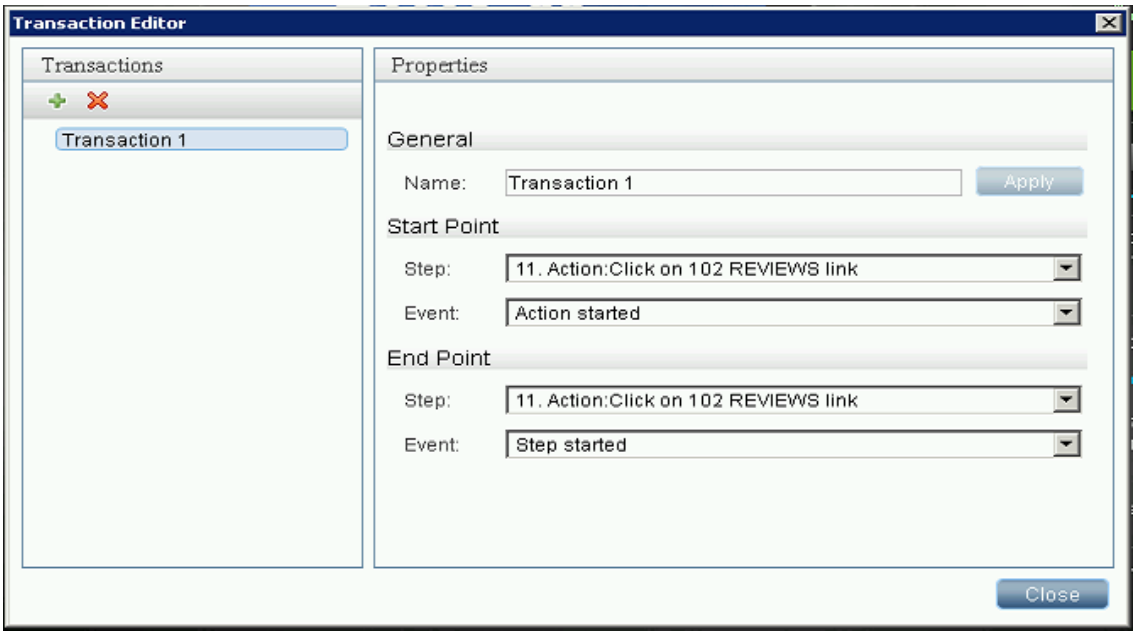
To access	Right click on a highlighted step in the script, and then select **Group Into > New Function**.
Relevant tasks	" How To Create and Use Function and Libraries" on page 442

User interface elements are described below:

UI Element	Description
Function Name	Enables you to specify a function name.
Add to library	• **Select Existing.** Save new function to an existing library. • **Create New.** Save new function to a new library.
Replace steps with function call	If selected, VuGen will automatically insert a call function in place of steps.

Transaction Editor Dialog Box (Ajax TruClient)

This dialog box enables you to manage transactions in Ajax TruClient Vuser scripts.



To access	Click on the from the **TruClient Sidebar**
Important information	If you add a wait step inside a transaction in a AjaxTruClient Script, the log will detail think time which is equal to wait time + pacing time + max(minimum time).

User interface elements are described below:

UI Element	Description
	Adds a new transaction or deletes the selected transaction.
General	Enables you to edit the name of the transaction.
Start Point	The step and the event within the step which indicates the start of the transaction.
End Point	The step and the event within the step which indicates the end of the transaction.

Citrix Protocol

Citrix Protocol - Overview

Citrix Vuser scripts emulate Citrix ICA protocol communication between a Citrix client and server. VuGen records all activity during the communication and creates a Vuser script.

When you perform actions on the remote server, VuGen generates functions that describe these actions. Each function begins with a **ctrx** prefix. These functions emulate the analog movements of the mouse and keyboard. In addition, the **ctrx** functions allow you to synchronize the replay of the actions, by waiting for specific windows to open.

VuGen also allows you to record a Citrix NFUSE session. With Citrix NFUSE, the client is installed, but your interface is a browser instead of a client interface. To record NFUSE sessions,

you must perform a multi-protocol recording for Citrix and Web Vusers. In multi-protocol mode, VuGen generates functions from both Citrix and Web protocols during recording.

In the following example, **ctrx_mouse_click** simulates a mouse click on the left button.

```
ctrx_mouse_click(44, 318, LEFT_BUTTON, 0, CTRX_LAST);
```

For more information about the syntax and parameters, see the Function Reference (**Help > Function Reference**).

Citrix Recording Tips

When recording a Citrix Vuser script, be sure to follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. If you plan to record a simple Citrix ICA session, use a single protocol script. When recording an NFUSE Web Access session, however, you must create a multi-protocol script for Citrix ICA and Web (HTML/HTTP), to enable the recording of both protocols.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting. For more information about recording into sections, see "Script Sections" on page 106.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start > All Programs > Microsoft Word**, be sure to click on the line **All Programs**.

Do not Resize Windows

Although VuGen supports the resizing of windows during recording the session, we recommend that you do not move or resize them while recording. To change the size or position of a window, double-click on the relevant **Sync on Window** step in the **Step Navigator** and modify the window's coordinates.

Make Sure Resolution Settings are Consistent

To insure successful bitmap synchronization, make sure that the resolution settings match. On the recording machine, check the settings of the ICA client, the Recording Options, and the Run Time settings. On the load generators, check the settings of the ICA client, and make sure that they are consistent between all load generators and recording machines. If there is an inconsistency between the resolutions, the server traffic increases in order to make the necessary adjustments.

Add Manual Synchronization Points

While waiting for an event during recording, such as the opening of an application, we recommend that you add manual synchronization points, such as **Sync on Bitmap** or **Sync on Text**. For details, see "Citrix - Automatic Synchronization" on page 465.

Disable Client Updates

Disable client updates when prompted by the Citrix client. This will prevent forward compatibility issues between VuGen and newer Citrix clients that were not yet tested.

Windows Style

For **Sync on Bitmap** steps, record windows in the "classic" windows style—not the XP style.

Change the Windows Style to "classic"

1. Click in the desktop area.
2. Select **Properties** from the right-click menu.
3. Click the Theme tab.
4. From the **Theme** list, select **Windows Classic**.
5. Click **OK**.

Citrix Replaying Tips

Wildcards

You can use wildcards (*) in defining window names. This is especially useful where the window name may change during replay, by its suffix or prefix.

In the following example, the title of the **Microsoft Internet Explorer** window was modified with a wildcard.

```
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0,
"Welcome to MSN.com - Microsoft Internet Explorer");ctrx_mouse_click
(573, 61, LEFT_BUTTON, 0,
"* - Microsoft Internet Explorer");
```

For more information, see the Function Reference (**Help > Function Reference**).

Set Initialization Quota

To prevent overloading by multiple Vusers while connecting, set an initialization quota of 4 to 10 Vusers (depending on the capacity of the server) or apply ramp-up initialization using the Scheduler.

Enable Think Time

For best results, do not disable think time in the run-time settings. Think time is especially relevant before the **ctrx_sync_on_window** and **ctrx_sync_on_bitmap** functions, which require time to stabilize.

Regenerate Script

During recording, VuGen saves all of the agent information together with the script. By default, it also includes this information in the script, excluding the **Sync On Text** steps. If you encounter text synchronization issues, then you can regenerate the script to include the text synchronization

steps.

In addition, if you disabled the generation of agent information in the Recording options, you can regenerate the script to include them.

Regenerating scripts is also useful for scripts that you manually modified. When you regenerate the script, VuGen discards all of your manual changes and reverts back to the originally recorded version.

To regenerate a script, select **Record > Regenerate Script** and select the desired options. For more information about regenerating scripts, see "How to Regenerate a Vuser Script" on page 113.

Set Consistency Between Machines

If you intend to replay the script on another machine, make sure that the following items are consistent between the record and replay machines: Window Size (resolution), Window Colors, System Font and the other Default Options settings for the Citrix client. These settings affect the hash value of bitmaps, and inconsistencies may cause replay to fail. To view the Citrix Client settings, select an item from the Citrix program group and select **Application Set Settings** or **Custom Connection Settings** from the right-click menu. Select the Default Options tab.

Increasing the Number of Vusers per Load Generator Machine

Load Generator machines running Citrix Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine, also known as the GDI (Graphics Device Interface). To increase the number of Vusers per machine, you can open a terminal server session on the machine which acts as an additional load generator.

The GDI count is Operating System dependent. The actual GDI (Graphics Device Interface) count for a heavily loaded machine using LoadRunner is approximately 7,500. The maximum available GDI on Windows 2000 machines is 16,384.

For more information on creating a terminal server session, see Terminal Services Overview.

Note: By default, sessions on a terminal server use a 256-color set. If you intend to use a terminal session for load testing, make sure to record on machines with a 256-color set.

Citrix Synchronization

Synchronization refers to waiting for windows and objects to become available before executing an action. This is necessary when recording Citrix scripts because, for example, if a step in a script opens a window, and the next step performs an action in that window, the second step cannot be implemented until the window opens. In order to ensure that VuGen does not replay the script incorrectly, it automatically generates functions that synchronize the script by waiting for windows or objects to become available. In addition, you can add synchronization functions manually.

For information about automatic synchronization, see "Citrix - Automatic Synchronization" on next page.

For information about manually adding synchronization points, see "How to Synchronize Citrix Scripts Manually" on page 471.

Citrix - Automatic Synchronization

During recording, VuGen automatically generates steps that help synchronize the Vuser's replay of the script:

Sync on Window

The **Sync On Window** step instructs the Vuser to wait for a specific event before resuming replay. The available events are **Create** and **Active**. The Create event waits until the window is created. The Active event waits until the window is created and then activated (in focus). Usually VuGen generates a function with a create event. If, however, the next instruction is a keyboard event, VuGen generates a function with an active event.

In the Editor, the corresponding function call to the **Sync On Window** step is **ctrx_sync_on_window**.

Sync on Obj Info

The **Sync On Obj Info** step instructs the Vuser to wait for a specific object property before resuming replay. The available attributes are **Enabled**, **Visible**, **Focused**, **Text**, **Checked**, **Lines**, or **Item**. The Enabled, Visible, Focused, and Checked attributes are boolean values that can receive the values **true** or **false**. The other attributes require a textual or numerical object value.

A primary objective of this step is to wait for an object to be in focus before performing an action upon it.

VuGen automatically generates **sync_on_obj_info** steps when the Citrix agent is installed and the Use Citrix Agent Input in Code Generation option is enabled in the Recording options. By default, this Recording option is enabled. For more information, see "Citrix > Code Generation" on page 268.

```
ctrx_sync_on_obj_info("Run=snapshot9", 120, 144, TEXT, "OK",
                                                 CTRX_LAST);
```

Sync on Text

The Text Synchronization step, **Sync On Text**, instructs the Vuser to wait for a text string to appear at the specified position before continuing. When replaying **Sync On Text**, Vusers search for the text in the rectangle whose modifiable coordinates are specified in the step's properties.

Caution: The maximum allowable length of a text string is 255 characters.

With an agent installation (see "Agent for Citrix Presentation Server - Overview" on page 467), you can instruct VuGen to automatically generate a text synchronization step before each mouse click or double-click. By default, automatic text synchronization is disabled. For more information, see "Citrix > Code Generation" on page 268.

Note, that even if you record a script with the option disabled, if you enable the option and regenerate the script, VuGen will insert text synchronization calls throughout the entire script.

In the Editor, the corresponding function call to the **Sync On Text** step is **ctrx_sync_on_text_ex**.

The following segment shows a **ctrx_sync_on_text_ex** function that was recorded during a Citrix recording with the HP Citrix Agent installed and text synchronization enabled.

```
ctrx_sync_on_window ("ICA Seamless Host Agent", ACTIVATE, 0, 0,391,
224, "snapshot1", CTRX_LAST);
ctrx_sync_on_text_ex (196, 198, 44, 14, "OK", "ICA Seamless Host
Agent=snapshot2", CTRX_LAST);
ctrx_obj_mouse_click ("<class=Button text=OK>", 196, 198, LEFT_BUTTON,
0, "ICA Seamless Host Agent=snapshot2", CTRX_LAST);
```

For more information on this function, see the Function Reference (**Help > Function Reference**).

See "Citrix Alternate Synchronization" below for additional information.

Citrix Alternate Synchronization

In addition to automatic synchronization in Citric Vuser scripts, you can add several other steps that affect the synchronization indirectly:

Setting the Waiting Time

The **Set Waiting Time** step sets a waiting time for the other Citrix synchronization functions. This setting applies to all functions that follow it within the script. For example, if your **Sync on Window** steps are timing out, you can increase the default timeout from 60 seconds to 180 seconds.

To insert this step, select **Insert > Add Step > Set Waiting Time**.

Checking if a Window Exists or Closed

The **ctrx_win_exist** step checks if a window is visible in the Citrix client. By adding control flow statements, you can use this function to check for a window that does not always open, such as a warning dialog box. In the following example, **ctrx_win_exist** checks whether a browser was launched. The second argument indicates how long to wait for the browser window to open. If it did not open in the specified time, it double-clicks its icon.

```
if (!ctrx_win_exist("Welcome",6, CTRX_LAST))
   ctrx_mouse_double_click(34, 325, LEFT_BUTTON, 0, CTRX_LAST)
```

To insert this step, select **Insert > Add Step > Win Exist**.

Another useful application for this step is to check if a window has been closed. If you need to wait for a window to close, you should use a synchronization step such as **UnSet Window** or **ctrx_ unset_window**.

For detailed information about these functions, see the Function Reference (**Help > Function Reference**).

Waiting for a Bitmap Change

In certain cases, you do not know what data or image will be displayed in an area, but you do expect it to change. To emulate this, you can use the **Sync on Bitmap Change** step or its corresponding function, **ctrx_sync_on_bitmap_change**. Right-click in the snapshot, and select **Insert Sync on Bitmap** from the right-click menu. VuGen inserts the step or function at the location of the cursor.

The syntax of the functions is as follows:

```
ctrx_sync_on_bitmap (x_start, y_start, width, height, hash, CTRX_
LAST);
ctrx_sync_on_bitmap_change (x_start, y_start, width, height,
                                          [initial_wait_time,]
```

```
[timeout,]                                                    [initial_
bitmap_value,] CTRX_LAST);
```

The following optional arguments are available for **ctrx_sync_on_bitmap_change**:

- initial wait time value—when to begin checking for a change.
- a timeout—the amount of time in seconds to wait for a change to occur before failing.
- initial bitmap value—the initial hash value of the bitmap. Vusers wait until the hash value is different from the specified initial bitmap value.

In the following example, the recorded function was modified and assigned an initial waiting time of 300 seconds and a timeout of 400 seconds.

```
ctrx_sync_on_bitmap_change(93, 227, 78, 52,
                           300,400,
"66de3122a58baade89e63698d1c0d5dfa",CTRX_LAST);
```

Note: If you are using **Sync on Bitmap**, make sure that the Configuration settings in the Controller, Load Generator machine, and screen are the same. Otherwise, VuGen may be unable to find the correct bitmaps during replay. For information on how to configure the client settings, see "Recording Options" on page 258.

Agent for Citrix Presentation Server - Overview

The Agent for Citrix Presentation Server, or Citrix Agent, is an optional utility that you can install on the Citrix server. It provides enhancements to the normal Citrix functionality. The following sections describe these enhancements.

It is provided in the product's installation disk and you can install it on any Citrix server. For more information about installing the Citrix Agent, see "How to Install and Uninstall the Citrix Agent" on page 472.

Object Detail Recording

When the Agent for Citrix Presentation Server is installed, VuGen records specific information about the active object instead of general information about the action. For example, VuGen generates **Obj Mouse Click** and **Obj Mouse Double Click** steps instead of **Mouse Click** and **Mouse Double Click** that it generates without the agent.

The following example shows the same mouse-click action recorded with and without the agent installation. Note that with an agent, VuGen generates ctrx_obj_xxx functions for all of the mouse actions, such as click, double-click and release.

```
/* WIthout Agent Installation */
ctrx_mouse_click(573, 61, LEFT_BUTTON, 0, test3.txt - Notepad);
/* WIth Agent Installation */
ctrx_obj_mouse_click("<text=test3.txt - Notepad class=Notepad>" 573,
         61, LEFT_BUTTON, 0, test3.txt - Notepad=snapshot21, CTRX_
LAST);
```

In the example above, the first argument of the **ctrx_obj_mouse_click** function contains the text of the window's title and the class, Notepad. Note that although the agent provides additional

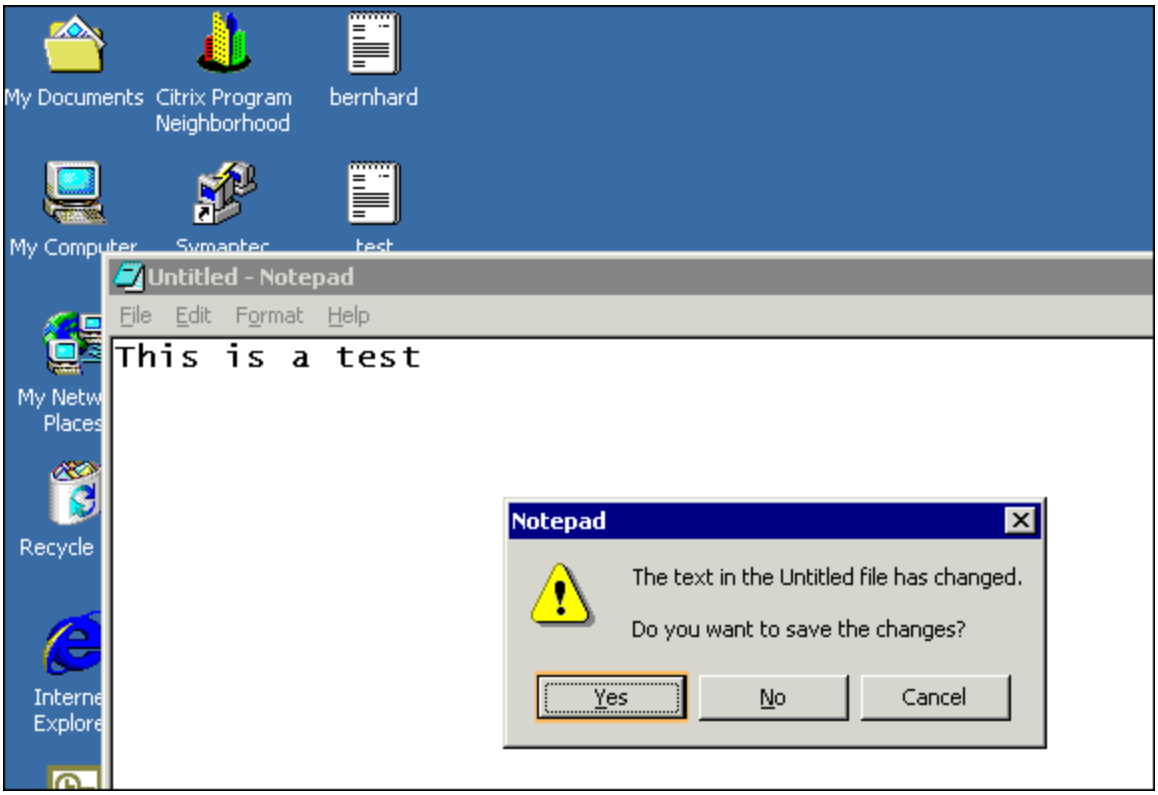
information about each object, Vusers only access objects by their window name and its coordinates.

Active Object Recognition

The agent installation lets you see which objects in the client window are detected by VuGen. This includes all Windows Basic Objects such as edit boxes, buttons, and item lists in the current window.

To see which objects were detected, you move your mouse through the snapshot. VuGen highlights the borders of the detected objects as the mouse passes over them.

In the following example, the **Yes** button is one of the detected objects.



Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When no agent is installed, you are limited to the **Insert Mouse Click**, **Insert Mouse Double Click**, **Insert Sync on Bitmap** and **Insert Get Bitmap Value**. If you are using a 256-color set, the **Insert Sync on Bitmap** and **Get Bitmap Value** steps are not available from the right-click menu.

When the Agent for Citrix Presentation Server is installed, the following additional options are available from the right-click menu of window in focus:

- **Obj Get Info** and **Sync on Obj Info**. Provide information about the state of the object: ENABLED, FOCUSED, VISIBLE, TEXT, CHECKED, and LINES.
- **Insert Sync on Obj Info**. Instructs VuGen to wait for a certain state before continuing. This is generated as a **ctrx_sync_on_obj_info** function.

- **Insert Obj Get Info**. Retrieves the current state of any object property. This is generated as a **ctrx_get_obj_info** function.
- **Insert Sync on Text** and **Get Text**. These are discussed in the section "Agent for Citrix Presentation Server - Overview" on page 467.

These commands are interactive—when you insert them into the script, you mark the object or text area in the snapshot.

In the following example, the **ctrx_sync_on_obj_info** function provides synchronization by waiting for the Font dialog box to come into focus.

```
ctrx_sync_on_obj_info("Font", 31, 59, FOCUSED, "TRUE", CTRX_LAST);
```

Utilizing VuGen's ability to detect objects, you can perform actions on specific objects interactively, from within the snapshot.

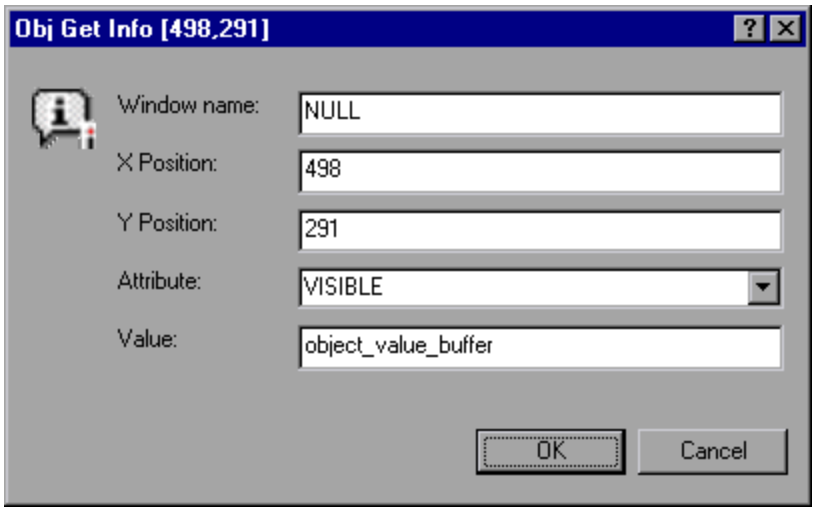
Insert a Function Interactively Using the Agent Capabilities

1. Click at a point within the Step Navigator to insert the new step. Make sure that a snapshot is visible.
2. Click within the snapshot.
3. To mark a bitmap, right-click on it and select **Insert Sync on Bitmap**.

   VuGen issues a message indicating that you need to mark the desired area by dragging the cursor. Click **OK** and drag the cursor diagonally across the bitmap that you want to select.

   When you release the mouse, VuGen inserts the step into the script after the currently selected step.
4. For all other steps, move your mouse over snapshot objects to determine which items are active—VuGen highlights the borders of active objects as the mouse passes over them.

   Right-click and select one of the Insert commands. A dialog box opens with the step's properties.



   Set the desired properties and click **OK**. VuGen inserts the step into your script.

Text Retrieval

With the agent installed, VuGen lets you save standard text to a buffer. Note that VuGen can only save true text—not a graphical representation of text in the form of an image.

You save the text using the **Sync On Text** step either during or after recording.

For task details, see "How to Synchronize Citrix Scripts Manually" on next page.

Troubleshooting XenApp 5.0

When recording with XenApp 5.0 using a Citrix and Web multi-protocol script, manual modifications are needed to ensure proper recording.

Record using XenApp 5.0

1. **Modify XenApp Settings:**
   a. Connect to the XenApp server through a web interface.
   b. Select **Preferences > Session Settings** and set **Window Size** to **No Preference**.
2. **Update Correlation Rules**
3. **Modify Recording Options**
   a. Open the **General > Recording** node of the Recording Options dialog box.
   b. Select **Advanced HTML** to open the Advanced HTML dialog box.
   c. Select **A script containing explicit URL's only**.
4. **Manually Modify Script**

How to Configure the Citrix Client and Server

Before creating a script, make sure you have a supported Citrix client installed on your machine, and that your server is properly configured.

Configure the Citrix client

In order to run your script, you must install a Citrix client on each Load Generator machine. If you do not have a client installed, you can download one from the Citrix website www.citrix.com under the **download** section.

VuGen supports all Citrix clients with the exception of versions 8.00, version 6.30.1060 and earlier, and Citrix Web clients.

Install the MetaFrame server

Make sure the MetaFrame server (3, 4, or 4.5) is installed. To check the version of the server, select **Citrix Connection Configuration** on the server's console toolbar and select **Help > About**.

Configure the MetaFrame server

Configure the Citrix server to completely close a session. After a Citrix client closes the connection, the server is configured, by default, to save the session for the next time that client opens a new connection. Consequently, a new connection by the same client will face the same

workspace from which it disconnected previously. It is preferable to allow each new test run to use a clean workspace.

The make sure that you have a clean workspace for each test, you must configure the Citrix server not to save the previous session. Instead, it should reset the connection by disconnecting from the client each time the client times-out or breaks the connection.

Configure the server

1. Open the Citrix Connection Configuration dialog box. Select **All Programs > Citrix > Administration Tools > Citrix Connection Configuration Tool**.
2. Select the ica-tcp connection name and select **Connection > Edit**. Alternatively, double-click on the connection. The Edit Connection dialog box opens.
3. Click **Advanced**. The Advanced Connection Settings dialog box opens.
4. In the bottom section of the dialog box, clear the **inherit user config** check box adjacent to the **On a broken or timed-out connection** list. Change the entry for this list box to **reset**.

How to Synchronize Citrix Scripts Manually

In addition to the automatic synchronization, you can manually add synchronization both during and after recording. A common use of this capability is where the actual window did not change, but an object within the window changed. Since the window did not change, VuGen did not detect or record a **Sync on Window**.

For example, if you want the replay to wait for a specific graphic image in a browser window, you insert manual synchronization. Or, if you are recording a large window with several tabs, you can insert a synchronization step to wait for the new tab's content to open.

Synchronize manually during recording

To add synchronization during recording, you use the floating toolbar. The **Sync On Bitmap** and **Sync on Text** functions lets you to mark an area or text within the client window that needs to be in focus before resuming replay.

- To insert a **Sync on Bitmap** step, click the **InsertSync on Bitmap**  button on the toolbar and mark a rectangle around the desired area.
- To insert a **Synch on Text** step (Citrix Agent required), click the **Insert Sync On Text**  button on the toolbar and mark a rectangle around the desired text.

Synchronize manually after recording

You can also add synchronization after the recording session. To add a synchronization step, right-click in the Snapshot pane, and select a synchronization option:

- **Sync on Bitmap.** Waits until a bitmap appears
- **Sync on Obj Info.** Waits until an object's attributes have the specified values (agent installations only)
- **Sync on Text.** Waits until the specified text is displayed (agent installations only)

How to Install and Uninstall the Citrix Agent

The installation file for the Agent for Citrix Presentation Server is located on the LoadRunner installation disk, under the **Additional Components\ Agent for Citrix Presentation Server** folder.

Note that the agent should be installed on your Citrix server machine only—not on Load Generator machines.

Install the Agent for Citrix Presentation Server

1. If you are upgrading the agent, make sure to uninstall the previous version before installing the next one.
2. If your server requires administrator permissions to install software, log in as an administrator to the server.
3. If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

   ```
   Change user /install
   ```

4. Locate the installation file, **Setup.exe**, on the product installation disk in the **Additional Components\ Agent for Citrix Presentation Server\Win32 or Win64** folder.
5. Follow the installation wizard to completion.

Note: After installation the agent will only be active for LoadRunner invoked Citrix sessions—it will not be active for users who start a Citrix session without LoadRunner.

Uninstall the Agent for Citrix Presentation Server

1. If your server requires administrator privileges to remove software, log in as an administrator to the server.
2. Open **Add/Remove Programs** in the server machine's Control Panel. Select **HP Software Agent for Citrix Presentation Server 32 or 64** and click **Change/Remove**.

Citrix Functions

During a Citrix recording session, VuGen generates functions that emulate the communication between a client and a remote server. The generated functions have a **ctrx** prefix. For example, **ctrx_obj_mouse_click** emulates a mouse click for a specific object. You can manually edit or add any of the functions into your Vuser script after the recording session.

For more information about the **ctrx** functions, see the Function Reference (**Help > Function Reference**).

Note that for the functions that specify a window name, you can use the wildcard symbol, an asterisk (*). You can place the wildcard at the beginning, middle, or end of the string.

Citrix - Understanding ICA Files

Citrix ICA client files are text files that contain configuration information for the applications accessed through the Citrix client. These files must have an .ica extension and must conform to the following format:

```
[WFClient]
Version=
TcpBrowserAddress=

[ApplicationServers]
AppName1=

[AppName1]
Address=
InitialProgram=#
ClientAudio=
AudioBandwidthLimit=
Compress=
DesiredHRES=
DesiredVRES=
DesiredColor=
TransportDriver=
WinStationDriver=
Username=
Domain=
ClearPassword=
```

Note: When you load an ICA file using the Recording Options, VuGen saves the file together with your script, eliminating the need to copy the ICA file to each load generator machine.

The following example shows a sample ICA file for using Microsoft Word on a remote machine through the Citrix client:

```
[WFClient]
Version=2
TcpBrowserAddress=235.119.93.56

[ApplicationServers]
Word=

[Word]
Address=Word
InitialProgram=#Word
ClientAudio=On
AudioBandwidthLimit=2
Compress=On
DesiredHRES=800
DesiredVRES=600
DesiredColor=2
```

```
TransportDriver=TCP/IP
WinStationDriver=ICA 3.0
Username=test
Domain=user_lab
ClearPassword=test
```

For more information, see the Citrix website www.citrix.com.

Failed Bitmap Synchronization Dialog Box

This dialog box enables you to decide what to do when a bitmap synchronization fails.

To access	This dialog box opens automatically during replay when there is a mismatch between the record snapshot and the replay snapshot in a bitmap synchronization step.

User interface elements are described below:

UI Element	Description
Continue	Accept the mismatch and use both the original and new snapshots as a basis for comparison between screens during future replays. If replay returns either one of the bitmaps, the Vuser will not fail.
Recording Snapshot	A view of the recording snapshot.
Replay Snapshot	A view of the replay snapshot.
Stop	Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution by default. Alternatively, you can specify **Continue on Error** for a specific function as described in "Citrix - Troubleshooting and Limitations" below.

Citrix - Troubleshooting and Limitations

This section describes troubleshooting and limitations for Citrix Vusers.

Effects and Memory Requirements of Citrix Agent

When you run Citrix Vusers with the agent installed, each Vuser runs its own process of **ctrxagent.exe**. This results in a slight reduction in the number of Vusers that can run on the server machine (about 7%).

The memory requirements per Citrix ICA Vuser (each Vuser runs its own **ctrxagent.exe** process) is approximately 4.35 MB. To run 25 Vusers, you would need 110 MBs of memory.

Data Execution Prevention (DEP) and Citrix Agent Performance

DEP is a security feature included in Windows version XP Service Pack 2 and later. DEP can interfere with some functions of the Agent for Citrix Presentation Server and may cause VuGen to stop recording.

If you experience unusual behavior during recording in these environments, modify the DEP settings.

Modify the Windows DEP settings

1. Open **Start > Control Panel > System**.
2. In the Advanced tab, click **Performance settings**.
3. In the Performance Options Data Execution Prevention tab, select the first option, **DEP for essential services only**.

   If you cannot change this option, click **Add**. Browse to the client program, for example IEXPLORE.EXE.

   If neither of these options are possible, try to disable DEP completely.

   a. In the Control Panel, click the **Advanced** tab under **System** section.

   b. Under **Startup and Recovery**, click **Edit**.

   c. Replace **NoExecute=OptOut** with **NoExecute=AlwaysOff**.
4. Click **OK** to save the settings.
5. Reboot the machine.

Debugging Tips

The following section lists debugging tips for Citrix scripts.

Single Client Installation

If you are unsuccessful in recording any actions in your Citrix session, verify that you have only one Citrix client installed on your machine. To verify that only one client is installed, open the Add/Remove Programs dialog box from the Control Panel and make sure that there is only one entry for the Citrix ICA client.

Add Breakpoints

Add breakpoints to your script in VuGen to help you determine the problematic lines of code.

Synchronize Your Script

If replay fails, you may need to insert synchronization functions into your script to allow more time for the desired windows to come into focus. Although you can manually add a delay using **lr_think_time**, we recommend that you use one of the synchronization functions discussed in "Citrix - Automatic Synchronization" on page 465.

Continuing on Error

You can instruct Vusers to continue running even after encountering an error, such as not locating a matching window. You specify Continue on Error for individual steps.

This is especially useful where you know that one of two windows may open, but you are unsure of which. Both windows are legal, but only one will open.

Indicate Continue on Error:

In the **Step Navigator**, right-click on the step and select **Show Arguments**. In the **Continue on Error** box, select the **CONTINUE_ON_ERROR** option.

In **Script view**, locate the function and add **CONTINUE_ON_ERROR** as a final argument, before **CTRX_LAST**.

This option is not available for the following functions: **ctrx_key**, **ctrx_key_down**, **ctrx_key_up**, **ctrx_type**, **ctrx_set_waiting_time**, **ctrx_save_bitmap**, **ctrx_execute_on_window**, and **ctrx_set_exception**.

Extended Log

You can view additional replay information in the Extended log. To do this, enable Extended logging in the Run-Time settings (f4 Shortcut key) **Log** tab. You can view this information in the Replay Log tab or in the output.txt file in the script's folder.

Snapshot Bitmap

When an error occurs, VuGen saves a snapshot of the screen to the script's **output** folder. You can view the bitmap to try to determine why the error occurred.

During recording, the bitmaps generated for the **ctrx_sync_on_bitmap** function are saved under the script's **data** folder. The bitmap name has the format of **hash_value.bmp**. If synchronization fails during replay, the generated bitmap is written to the script's output folder, or if you are running it in a scenario, to wherever the output files are written. You can examine the new bitmap to determine why synchronization failed.

Show Vusers

To show Vusers during a scenario, enter the following in the Vuser command line box: **-lr_citrix_vuser_view**. In the Controller, open the Vuser Details dialog box and click **More** to expand the dialog box. Note that this will affect the scalability of the test, so this should only be done to examine a problematic Vuser's behavior.

To reduce the effect on the script's scalability, you can show the details for an individual Vuser by adding the Vuser's ID at the end of the command line: `-lr_citrix_vuser_view <VuserID>`.

To open multiple Vusers, place a comma-separated list of IDs after the command line. Do not use spaces, but you may use commas or dashes. For example, 1,3-5,7 would show Vusers 1,3,4,5, and 7, but would not show Vuser 2, 6 or any Vuser with an ID higher than 7.

When recording with XenApp 5.0 using a Citrix and Web multi-protocol script, manual modifications are needed to ensure proper recording.

Record using XenApp 5.0

1. **Modify XenApp Settings**
   a. Connect to the XenApp server through Internet Explorer.
   b. Log in using the same account that will be used during recording.
   c. Select **Preferences > Session Settings** and set **Window Size** to **No Preference**.
2. **Update Correlation Rules**
   a. Open the **HTTP Correlation** node of the Recording Options dialog box.
   b. Make sure that the **Citrix_XenApp** rule is selected.
3. **Modify Recording Options**

a. Open the **General > Recording** node of the Recording Options dialog box.

b. Select **Advanced HTML** to open the Advanced HTML dialog box.

c. Select **A script containing explicit URL's only**.

Load Generator and Citrix XenApp 11.2 Client

Load Generator cannot be used in service mode for the Citrix protocol when the Citrix XenApp client 11.2 is used. A workaround is to use another version of Citrix client.

Citrix Access Gateway Support

LoadRunner supports Citrix Access Gateway (CAG) for Citrix Client versions 10.200 (or lower) and Citrix Client version 13.x.

COM Protocol

COM Protocol Overview

When you record COM client applications, VuGen generates functions that describe COM client-server activity. The recorded script contains interface declarations, API calls and instance calls to methods. Each COM function begins with an **lrc** prefix. You can configure the programming language in which to create a Vuser script as either C or Visual Basic.

For each COM Vuser script, VuGen creates the following:

- Interface pointer and other variable declarations in the interfaces.h file.
- Function calls that you can record in the vuser_init, actions or vuser_end sections of the Vuser file.
- A user.h file containing the translation of the Vuser script into low level calls.

COM Technology Overview

This section provides an outline of COM technology. This should be enough to get you started with COM Vuser scripts. See Microsoft Developer's Network (MSDN) and other documentation for further details.

COM (Component Object Model) is a technology for developing reusable software components ("plug-ins"). DCOM (Distributed COM) allows use of COM components on remote computers. Microsoft transaction servers (MTS), Visual Basic and Explorer all use COM/DCOM technology. Thus, the application you are testing may use COM technology indirectly, even though you don't know it. You will probably have to include some, but certainly not all, of the COM calls made by your application in the Vuser script.

Objects, Interfaces and Type Libraries

COM objects are binary code modules. Each COM object implements one or more interfaces that allow client programs to communicate with it. You need to know about these interfaces in order to follow the COM calls in the Vuser scripts. Type libraries, used as a reference for accessing COM interface methods and parameters, contain descriptions of COM objects and interfaces. Each COM class, interface, and type library is identified by a Global Unique Identifier (GUID).

COM Interfaces

A COM interface provides a grouped collection of related methods. For example, a **Clock** object may have **Clock**, **Alarm** and **Timer** interfaces. Each interface has one or more methods. For example the **Alarm** interface may have **AlarmOn** and **AlarmOff** methods.

An interface may also have one or more properties. Sometimes, the same function may be performed by calling a method or by setting or getting the value of a property. For example, you can set the **Alarm Status** property to **On** or call the **AlarmOn** method.

A COM object may support many interfaces. The **IUnknown** interface is implemented by all components and is used to find out about other interfaces. Many components also implement the **IDispatch** interface, which exposes all other interfaces and methods of the object, allowing implementation of COM automation in scripting languages.

COM Class Context and Location Transparency

COM objects can run on the same machine as the client application, or on a remote server. COM objects that an application creates may be in a local library, a local process or a remote machine ("Remote Object Proxy"). The location of the COM object, known as the "Context," can be transparent to the application. Most users apply the Vusers to check the load on remote servers. Therefore, objects accessed by Remote Object Proxy are usually the most relevant for these tests.

COM Data Types

COM also provides several special data types, including safe arrays, BSTR strings and variants. You may need to use these data types for debugging, parameterization and similar tasks.

COM Vuser Script Structure

VuGen COM Vuser scripts are structured in a special way to meet the needs of COM interfaces.

Interface Methods

Calls to interface methods have the following names and syntax conventions:

```
lrc_<interface name>_<method name>(instance,...);
```

Note that the instance is always the first parameter passed.

The vendors of the respective COM components usually supply documentation for the interface functions.

Interface Pointers

The interface header file defines the interface pointers, as well as other variables, that can be used in the script. Each interface has an Interface ID (IID) which uniquely identifies the interface.

The format of the interface definition is:

```
<interface type>*<interface name> = 0; //"{<IID of the interface
type>}"
```

In the following example, the interface type is IDispatch, the name of the interface instance is IDispatch_0, and the IID of IDispatch type is the long number string:

```
IDispatch* IDispatch_0= 0;//"{00020400-0000--C000-000000000046}"
```

Vuser Script Statements

The COM Vuser script consist of code that creates object instances, retrieves interface pointers and calls the interface methods. Each user action may generate one or more COM calls. Each COM call is coded by VuGen as a group of statements. Each such group is contained in a separate scope enclosed in braces. Several different statements prepare for the main call by assigning values and performing type conversions. For example, the group of calls needed to create an object may look like this:

```
{
GUID pClsid = lrc_GUID("student..1");
IUnknown * pUnkOuter = (IUnknown*)NULL;
unsigned long dwClsContext = lrc_ulong("7");
GUID riid = IID_IUnknown;
lrc_CoCreateInstance(=;pClsid, pUnkOuter, dwClsContext, =;riid,
(void**)=;IUnknown_0, CHECK_HRES);
}
```

Error Checking

Each COM method or API call returns an error value. VuGen will set a flag to check or not to check errors during replay, depending upon whether the call succeeded during the original recording. The flag appears as the last argument of the function call and has these values:

CHECK_HRES	This value is inserted if the function passed during recording and errors should be checked during replay.
DONT_ CHECK_HRES	This value is inserted if the function failed during recording and errors should not be checked during replay.

COM Sample Vuser Scripts

This section shows examples of how VuGen emulates a COM client application. It is divided up into the basic COM script operations. Each type of operation is done within a separate scope.

Instantiation of the Object

To use a COM object, the application must first instantiate it and get a pointer to an interface of that object.

VuGen does the following to instantiate an object

1. VuGen calls lrc_GUID to get a unique ProgID for the object, to be stored in pClsid:

   ```
   GUID pClsid = lrc_GUID("student..1");
   ```

   pClsid is the unique global CLSID of the object, which was converted from the **student.student.1** ProgID.

2. If the unknown interface pointer is a pointer to an aggregated object, VuGen retrieves the pointer to that object, or else it sets it to NULL:

   ```
   IUnknown * pUnkOuter = (IUnknown*)NULL;
   ```

3. VuGen sets the contexts of the object to be created:

```
unsigned long dwClsContext = lrc_ulong("7");
```

dwClsContext contains the context of the object (in process, local, remote or combinations of these.)

4. VuGen sets a variable to hold the requested interface ID, which is IUnknown in this case:

```
GUID riid = IID_IUnknown;
```

riid contains the interface ID of the **IUnknown** interface.

5. After the input parameters are prepared, a call to lrc_CoCreateInstance creates an object using the parameters defined in the preceding statements. A pointer to the IUnknown interface is assigned to output parameter IUnknown_0. This pointer is needed for subsequent calls:

```
lrc_CoCreateInstance(=;pClsid, pUnkOuter, dwClsContext, =;riid,
(void**)=;IUnknown_0, CHECK_HRES);
```

The input parameters were prepared and explained above. Since the call succeeded, VuGen sets error checking on during the user simulation by inserting the **CHECK_HRES** value. The call returns a pointer to the IUnknown interface in **IUnknown_0**, that can be used in subsequent calls.

Retrieving an Interface

After creating an object, VuGen has access only to the **IUnknown** interface . VuGen will use the **IUnknown** interface for communicating with the object. This is done using the **QueryInterface** method of the **IUnknown** standard interface. The first parameter in a VuGen method call is the interface instance. In this case it is the **IUnknown_0** pointer set previously by **CoCreateInstance**. The **QueryInterface** call requires as input the ID of the interface to be retrieved, and returns a pointer to the interface designated by that ID.

Get the Interface

1. First, VuGen sets a parameter, riid, equal to the ID of the Istudent interface:

```
GUID riid = IID_Istudent;
```

2. A call to QueryInterface assigns a pointer to the Istudent interface to output parameter Istudent_0 if the Istudent object has such an interface:

```
lrc_IUnknown_QueryInterface(IUnknown_0, =;riid, (void**)=;Istudent_
0, CHECK_HRES);
```

Using an Interface to Set Data

Here is an example of using the methods of the interface to set data. Suppose that in the application, the user is supposed to input a name. This activates a method for setting the name. VuGen records this in two statements. One statement is used for setting up the name string and the second one sets the name property.

Set up the Entire Function Call

1. First, VuGen sets a variable (Prop Value) equal to the string. The parameter is of type BSTR, a string type used in COM files:

```
BSTR PropValue = lrc_BSTR("John Smith");
```

In subsequent stages, you will probably parameterize this call, replacing "John Smith" with a parameter, so that different names are used each time the Vuser script is run.

2. Next, VuGen calls the Put_Name method of the Istudent interface to enter the name:

```
lrc_Istudent_put_name(Istudent_0, PropValue, CHECK_HRES);
```

Using an Interface to Return Data

Returning data from an application is different than entering the data, because you might want to store these values and use them as inputs in subsequent calls for parameterization.

The following is an example of what VuGen may do when the application retrieves data

1. Create a variable of the appropriate type (in this case a BSTR) that will contain the value of the property.

```
BSTR pVal;
```

2. Get the value of the property, in this case a name, into the **pVal** variable created above, using the get_name method of the **Istudent** interface in this example.

```
lrc_Istudent_get_name(Istudent_0, =;pVal, CHECK_HRES);
```

3. VuGen then generates a statement for saving the values.

```
//lrc_save_BSTR("param-name",pVal);
```

The statement is commented out. You can remove the comments and change <param-name> to a variable with a meaningful name to be used for storing this value. VuGen will use the variable to save the value of **pVal** returned by the previous call. You can then use the variable as a parameterized input in subsequent calls to other methods.

The IDispatch Interface

Most COM objects have specific interfaces. Many of them also implement a general-purpose interface called **IDispatch**, which VuGen translates in a special way. IDispatch is a "superinterface" that exposes all of the other interfaces and methods of a COM object. Calls to the **IDispatch:Invoke** method from VuGen scripts are implemented using **lrc_Disp** functions. These calls are constructed somewhat differently from calls to other interfaces.

The **IDispatch** interface **Invoke** method can execute a method, it can get a property value, or it can set a value or reference value for a property. In the standard **IDispatch:Invoke** method these different uses are signaled in a **wflags** parameter. In the VuGen implementation they are implemented in different procedure calls that invoke a method or put or get a property.

For example, a call to IDispatch to activate the GetAgentsArray method may look like this:

```
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray",
/*locale*/1033, LAST_ARG, CHECK_HRES);
```

The parameters in the above call are:

IDispatch_0	This is the pointer to the **IDispatch** interface returned by a previous call to the **IUnknown:Queryinterface** method.

GetAgentsArray	This is the name of the method to invoke. Behind the scenes, VuGen will get the ID of the method from the name.
1033	This is the language locale.
LAST_ARG	This is a flag to tell the **IDispatch** interface that there are no more arguments.
CHECK_HRES	This flag turns on checking of HRES, since the call succeeded when it was recorded.

In addition, there might be another parameter, OPTIONAL_ARGS. This signals that in addition to any standard parameters, VuGen is sending some optional arguments. Each optional argument consists of a pair giving the ID or name of the argument and its value. For example, the following call to lrc_DispMethod passes optional arguments "#3" and "var3":

```
{
        GUID riid = IID_IDispatch;
        lrc_IOptional_QueryInterface(IOptional_0, =;riid, (void**)
=;IOptional_0, CHECK_HRES);
    }
    {
        VARIANT P1 = lrc_variant_short("47");
        VARIANT P2 = lrc_variant_short("37");
        VARIANT P3 = lrc_variant_date("3/19/1901");
        VARIANT var3 = lrc_variant_scode("4");
        lrc_DispMethod((IDispatch*)IOptional_0, "in_out_optional_
args", /*locale*/1024, =;P1, =;P2, OPTIONAL_ARGS, "#3", =;P3, "var3",
=;var3, LAST_ARG, CHECK_HRES);
}
```

The different **lrc_Disp** methods that use the **IDispatch** interface are detailed in the .

Type Conversions and Data Extraction

As shown in the above example, many COM parameters are defined as variants. To extract these values, VuGen uses a number of conversion functions, derived from the equivalent COM functions. The full list is given in the . Previously, we showed how the **lrc_DispMethod1** call was used to retrieve an array of name strings:

```
VARIANT retValue = lrc_variant_empty();
retValue = lrc_DispMethod1((IDispatch*)IDispatch_0, "GetAgentsArray",
/*locale*/1033, LAST_ARG, CHECK_HRES);
```

The following example now shows how VuGen gets the strings out of **retValue**, which is a variant that will be read as an array of strings.

First, VuGen extracts the BSTR array from the variant:

```
BstrArray array0 = 0;
array0 = lrc_GetBstrArrayFromVariant(=;retValue);
```

With all the values in array0, VuGen provides you with code that you can use to extract the elements from the array for later use in parameterization, as in the example below:

```
//GetElementFrom1DBstrArray(array0, 0); // value: Alex
//GetElementFrom1DBstrArray(array0, 1); // value: Amanda
....
```

VuGen has numerous type conversion functions and functions for extracting conventional types from variants. These are detailed in the Function Reference (**Help > Function Reference**).

Selecting COM Objects to Record

The application you are testing may use a great many COM objects. Only a few may actually create load and may be important for the load test. Thus, before you record a COM application, you should select the objects you want to record for the load test. VuGen allows you to browse for objects from type libraries that it can read on the local machine and on other computers in the network.

Deciding Which Objects to Use

There are several ways to decide which COM objects should be included in the test. Try to determine which remote objects are used by the software. If you are unsure which objects to use, try using the default filter. The Environments branch of the filter includes calls to three sets of objects (ADO, RDS and Remote) that are likely to generate load on remote servers.

You can also check the actual calls to refine the filter. After you have recorded the test, you can save the file and look in the **data** folder that VuGen creates for a file named **lrc_debug_list_<nnn>.log"**, where **nnn** is the process number. This log file contains a listing of each COM object that was called by the recorded application, regardless of whether or not the recording filter included that object. Only calls that generate load on the server should be included for recording.

For example, the following is a local COM of the Visual Basic library:

```
Class JetES {039EA4C0-E696-11D0-878A-00A0C91EC756}
was loaded from type library "JET Expression Service Type Library"
({2358C810-62BA-11D1-B3DB-00600832C573} ver 4.0)
```

It should not be added since it does not generate load on the server.

Likewise, since the OLE DB and Microsoft Windows Common Controls are local objects, the following are examples of classes and libraries that are not going to place any load on the server and should not be recorded:

```
Class DataLinks {2206CDB2-19C1-11D1-89E0-00C04FD7A829}
was loaded from type library "Microsoft OLE DB Service Component 1.0
Type Library"
({2206CEB0-19C1-11D1-89E0-00C04FD7A829} ver 1.0)
Class DataObject {2334D2B2-713E-11CF-8AE5-00AA00C00905}
was loaded from type library "Microsoft Windows Common Controls 6.0
(SP3)"
({831FDD16-0C5C-11D2-A9FC-0000F8754DA1} ver 2.0)
```

However, for example, a listing such as the following indicates a class that should be recorded since it does generate load on the server:

```
Class Order {B4CC7A90-1067-11D4-9939-00105ACECF9A}
was loaded from type library "FRS"
({B4CC7A8C-1067-11D4-9939-00105ACECF9A} ver 1.0)
```

Calls to classes of the **FRS** library, used for instance in the flight_sample that is installed with VuGen, use server capacity and should be recorded.

If a COM object itself calls other COM objects, all the calls will be listed in the type information log file. For example, every time the application calls an **FRS** class function, the **FRS** library calls the **ActiveX Data Object (ADO)** library. If several functions in such a chain are listed in a filter, VuGen records only the first call that initiates the chain. If you selected both **FRS** and **ADO** calls, only the **FRS** calls will be recorded.

On the other hand, if you select only the **ADO** library in the filter, then calls to the **ADO** library will be recorded. It is often easier to record the call to the first remote object in the chain. In some cases, however, an application may use methods from several different COM objects. If all of them use a single object that puts a load on the server, you could only record the final common object.

Which Objects Can be Selected

VuGen can only record objects if it can read their type libraries. If the type libraries were not installed in the system or VuGen cannot find them, the COM objects will not be listed in the Recording Options dialog box. If they are used by your application, VuGen will not be able to identify these objects and will identify them as **INoTypeInfo** in the files.

Which Interfaces Can be Excluded

For each object, the Recording Options dialog box will show you all interfaces that are listed in the Type Library, and allow you to specify inclusion or exclusion of each one. However, **ADO**, **RDS** and Remote Objects can be included in the filter as a group. The filter will not show the individual objects of those environments or their interfaces. Objects that you included from type libraries may also have interfaces that are not listed in the type library and therefore not shown in the Recording Options dialog. After generating a VuGen script, you can identify these interfaces in the script and get their GUID numbers from the interfaces.h file that VuGen generates. Using this information, you can exclude the interfaces as explained in "COM/DCOM > Filter Node" on page 270.

Database Protocols

Database Protocols Overview

Suppose that you have a database of customer information that is accessed by customer service personnel located throughout the country. You use Database Vusers to emulate the situation in which the database server services many requests for information. A Database Vuser could:

- Connect to the server
- Submit an SQL query
- Retrieve and process the information
- Disconnect from the server

You distribute several hundred Database Vusers among the available load generators, each Vuser accessing the database by using the server API. This enables you to measure the performance of your server under the load of many users.

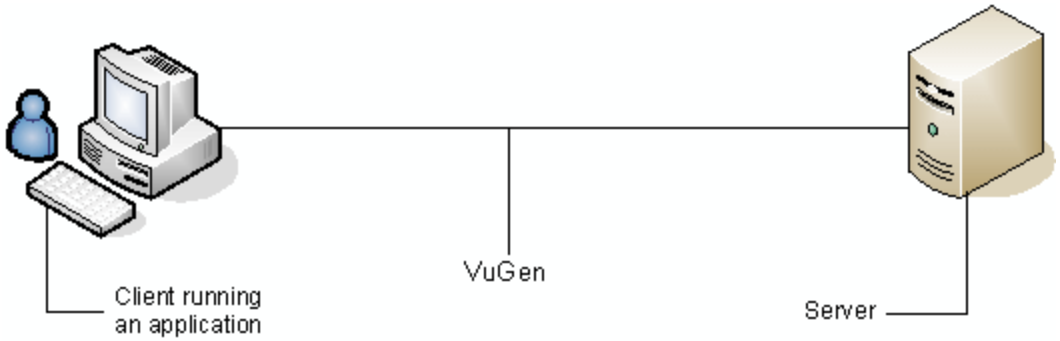
The program that contains the calls to the server API is called a Database Vuser script. It emulates the client application and all of the actions performed by it. The Vusers execute the script and

emulate user load on the client/server system. The Vusers generate performance data which you can analyze in report and graph format.

VuGen supports the following database types: CtLib, DbLib, Oracle, ODBC, and DB2-CLI. The resulting script contains LRD functions that describe the database activity.

VuGen Database Recording Technology

VuGen creates Database Vuser scripts by recording all the activity between a database client and a server. VuGen monitors the client end of the database and traces all the requests sent to and received from the database server.



Like all other Vusers created using VuGen, Database Vusers communicate with the server without relying on client software. Instead, each Database Vuser executes a script that executes calls directly to server API functions.

You create Database Vuser scripts in a Windows environment using VuGen. Once you create a script, you can assign it to Vusers in both Windows and Linux environments.

Users working in a Linux only environment can create Database Vuser scripts through programming using VuGen templates as the basis for a script. For information about programming Database Vuser scripts on Linux, see "Creating and Running Scripts in Linux" on page 825.

Database Grids

When you record or replay a Vuser script, the data that is returned by each query is displayed in a data grid. In a Vuser script, the existence of a data grid is indicated by a **GRID** statement. VuGen displays data grids in either the Data Grids pane or the Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

To correlate a value in a data grid:

Display the data grid in the Snapshot pane, right-click in a cell inside the data grid, and select **Create Correlation**.

To save the data in a data grid to a file:

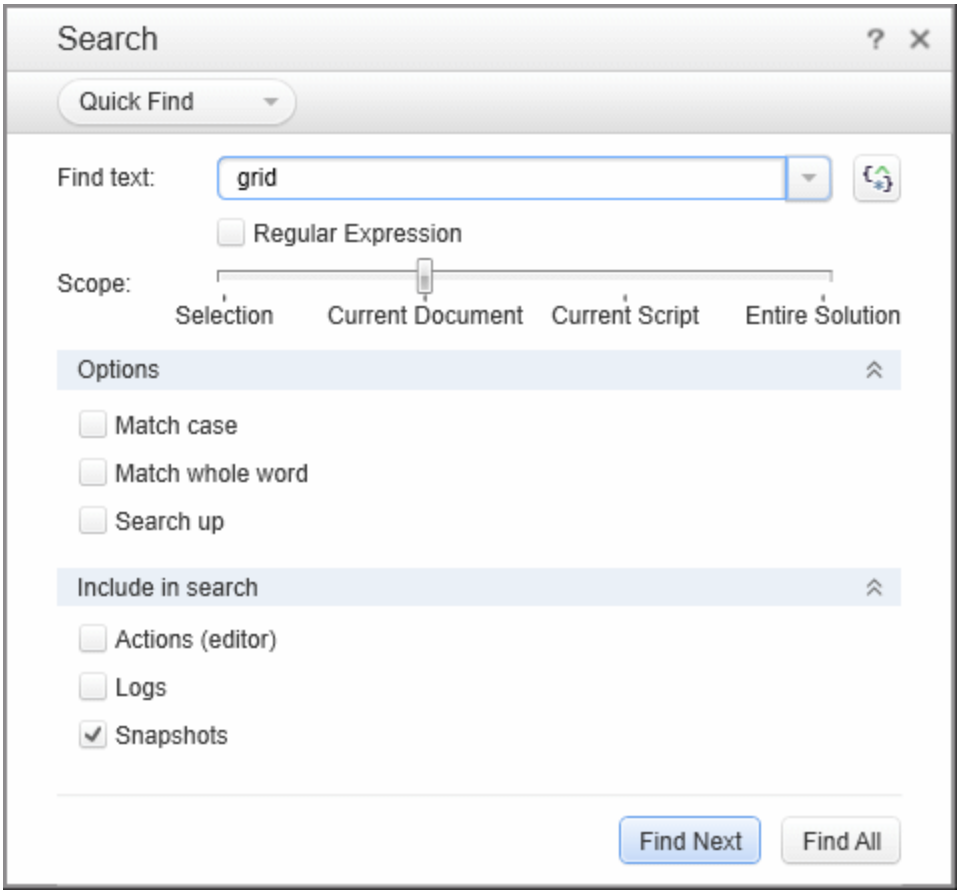
Display the data grid in the Snapshot pane or the Data Grids pane, right-click in any cell in the data grid, and select **Save Grid To File**.

To copy the text from a cell in a data grid to the clipboard:

Display the data grid in the Snapshot pane or in the Data Grids pane, right-click in the cell in the data grid, and select **Copy Selection**.

To search for data inside a data grid:

1. Display the data grid in the Snapshot pane, and click **Search > Quick Find** to open the Search dialog box.
2. Click **Include in Search**, and then select the **Snapshots** check box.



Oracle Applications

Oracle Applications is a two-tier ("fat" client) packaged application, made up of 35 different modules (Oracle Human Resources, Oracle Financials, and so forth).

There are a number of issues that you should be aware of while recording and replaying Vuser scripts for Oracle Applications:

- A typical script contains thousands of events, binds, and assigns.
- A typical script has many db connections per user session.
- Scripts almost always require correlated queries.
- Oracle Applications' clients are 16-bit only (developed with Oracle Developer 2000). This means that for debugging, if you don't have the Oracle 32bit client, you need to use VuGen's Force 16-bit options.

When a new window is created, the application retrieves an .xpf file from the file system for display. Currently, VuGen does not take this into consideration since it records at the client/server level. Therefore, there is a fairly significant inaccuracy in performance measurements since in most cases performance problems are related to the network bottleneck between clients and file server. We are currently thinking about this problem and how, if at all, to solve it.

Handling Database Errors

You can control how database Vusers handle errors when you run a database Vuser script. By default, if an error occurs during script execution, the script execution is terminated. To change the default behavior, you can instruct the Vuser to continue when an error occurs. You can apply this behavior in different ways as described below.

Globally Modifying Error Handling

You can change the way that Vusers handle errors by issuing an LRD_ON_ERROR_CONTINUE or LRD_ON_ERROR_EXIT statement. By default, a Vuser aborts the script execution when it encounters any type of error—database, parameter related, and so on. To change the default behavior, insert the following line into your script:

```
LRD_ON_ERROR_CONTINUE;
```

From this point on, the Vuser continues script execution, even when an error occurs.

You can also specify that the Vuser continue script execution when an error occurs only within a segment of the script. For example, the following code tells the Vuser to continue script execution even if an error occurs in the lrd_stmt or lrd_exec functions:

```
LRD_ON_ERROR_CONTINUE;
lrd_stmt(Csr1, "select..."...);
lrd_exec(...);
LRD_ON_ERROR_EXIT;
```

Use the LRD_ON_ERROR_CONTINUE statement with caution, as significant and severe errors may be missed.

Locally Modifying Error Handling

You can set error handling for a specific function by modifying the severity level. Functions such as **lrd_stmt** and **lrd_exec,** which perform database operations, use severity levels. The severity level is indicated by the function's final parameter, **miDBErrorSeverity**. This parameter tells the Vuser whether or not to continue script execution when a database error occurs (error code 2009). The default, 0, indicates that the Vuser should abort the script when an error occurs.

For example, if you reference a table that does not exist, the following database statement fails and the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1
/*Deferred*/,
        1 /*Dflt Ora Ver*/, 0);
```

To tell a Vuser to continue script execution, even when a database operation error occurs for that function, change the statement's severity parameter from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1
/*Deferred*/,
        1 /*Dflt Ora Ver*/, 1);
```

When the severity is set to 1 and a database error occurs, a warning is issued. Note that the severity level set for a particular function applies only to that function.

CtLib Result Set Errors

In CtLib recording, the application retrieves all of the available result sets after executing a statement. If the returned result set contains fetchable data, the application performs bind and fetch operations on the data as indicated in the following example:

```
lrd_stmt(Csr15, "select * from all_types", -1, 148, -99999, 0);
lrd_exec(Csr15, 0, 0, 0, 0, 0);
lrd_result_set(Csr15, 1 /*Succeed*/, 4040 /*Row*/, 0);
lrd_bind_col(Csr15, 1, =;tinyint_D41, 0, 0);
...
lrd_fetch(Csr15, -9, 0, 0, PrintRow3, 0);
```

If a result set does not contain fetchable data, bind and fetch operations cannot be performed.

When you parameterize your script, result data may become unfetchable (depending on the parameters). Therefore, a CtLib session that recorded bind and fetch operations for a particular statement, may not be able to run, if the new data is unfetchable. If you try to execute an **lrd_bind_col** or an **lrd_fetch** operation, an error will occur (LRDRET_E_NO_FETCHABLE_DATA — error code 2064) and the Vuser will terminate the script execution.

You can override the error by telling the Vuser to continue script execution when this type of error occurs. Insert the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_CONT;
```

To return to the default mode of terminating the script execution, type the following line into your script:

```
LRD_ON_FETCHABLE_SET_ERR_EXIT;
```

Use this option with caution, as significant and severe errors may be missed.

Debugging Database Applications

The following tips apply to database applications only (Oracle, ODBC, Ctlib):

Generating Debugging Information

Note: You can now set options to view most of the information described in this section using VuGen's user interface.

VuGen contains an inspector "engine." You can force VuGen recorder to create "inspector" output by editing *\WINDOWS_DIR\vugen.ini* as follows:

```
[LogMode]
EnableAscii=ASCII_LOG_ON
```

When this option is enabled, VuGen creates a file, vuser.asc in the Data folder at the end of the recording. Note that this option should be used for debugging purposes only. This output file can become very large (several MB) and have serious effects on machine performance and disk space.

For cases like ODBC-based applications, it is possible to configure the ODBC Administrator (located in the Windows Control Panel) to provide a similar trace output. Open the ODBC options, and select `Trace ODBC calls' to ON. Similarly the ODBC Developer Kit provides a Spy utility for call tracing.

To enable further debug information, add the following section to the \WINDOWS_DIR\vugen.ini file:

```
[INSPECTOR]
TRACE_LEVEL=3
TRACE_FILENAME=c:\tmp\sqltrace.txt
```

The file (sqltrace.txt) will include useful internal information regarding the hooking calls made during recording. The trace_level is between 1 and 3, with 3 representing the most detailed debug level. Note that in VuGen versions 5.02 and higher, you can set the trace level from the user interface.

Examining Compiler Information

You can view information about each stage of code generation, preprocessing and compilation to determine the source of any errors.

Code Generation Information

Look at the vuser.log file under the Data folder. This file, which contains a log of the code generation phase, is automatically created at the end of every lrd recording (i.e. all database protocols).

The following is an example of a log file:

```
lrd_init: OK
lrd_option: OK
lrd_option: OK
lrd_option: OK
Code generation successful
lrd_option: OK
lrd_end: OK
```

If any of the messages are not OK or successful, then a problem occurred during the code generation.

Preprocessing and Compilation Information

During runtime, VuGen displays information about both the preprocessing and compilation processes.

Database Protocols - Troubleshooting and Limitations

This section describes troubleshooting and limitations for database protocols.

Troubleshooting all database protocols

IE crashes when recording Oracle NCA/11i scripts

This can occur due to an incompatible dll file.

Replace the incompatible dll

1. Open the **C:\program files\oracle\JInitiator_1.3.1.18\bin\hotspot** folder.
2. Back up the **jvm.dll** file.
3. Delete the **jvm.dll** file and replace it with a different version of the file.

Evaluating Error Codes

When a Vuser executes an LRD function, the function generates a return code. A return code of 0 indicates that the function succeeded. For example, a return code of 0 indicates that another row is available from the result set. If an error occurs, the return code indicates the type of error. For example, a return code of 2014 indicates that an error occurred in the initialization.

There are four types of return codes, each represented by a numerical range:

Type of Return Code	Range
Informational	0 to 999
Warning	1000 to 1999
Error	2000 to 2999
Internal Error	5000 to 5999

For more detailed information on the return codes, see the Function Reference (**Help > Function Reference**).

You can evaluate the return code of an LRD function to determine if the function succeeded. The following script segment evaluates the return code of an lrd_fetch function:

```
static int rc;
rc=lrd_fetch(Csr15, -13, 0, 0, PrintRow4, 0);
```

```
if (rc==0)
    lr_output_message("The function succeeded");
else
    lr_output_message("The function returned an error code:%d",rc);
```

Two-tier Database Scripting Tips

The following section offers solutions for two-tier database scripts.

Question 1: Why does the script fail when it is data driven, while the same values work with the application itself?

Answer: The failure may be a result of trailing spaces in your data values. Even though the data values that you type directly into the GUI are probably truncated, you should manually eliminate them from your data file. Tab-delimited files can hide trailing spaces and therefore obscure problems. In general, comma-delimited files are recommended. You can view the files in Excel to see if things are correct.

Question 2: Why does an SQL error of an invalid cursor state occur on the second iteration?

Answer: The **lrd_close_cursor** function may not have been generated or it may be in the *end* section instead of the *action* section. You will need to add a cursor close function or move it from the *end* section to make the script iterate successfully.

Opening a new cursor may be costly in terms of resources. Therefore, we recommend that you only open a cursor once in the *actions* section during the first iteration. You can then add a new parameter that contains the iteration number as a string by using the Iteration Number type. Call this parameter *IterationNum*. Then, inside the *actions* section replace a call to open a new cursor, for example,

lrd_open_cursor(=;Csr1, Con1, 0);

with

if (!strcmp(lr_eval_string("<IterationNum>"), "1"))

lrd_open_cursor(=;Csr1, Con1, 0);

Question 3: How can I fix code produced by VuGen that will not compile because of data declarations in the *vdf.h* file?

Answer: The problem, most likely, is an SQL data type that is not supported by VuGen. For Microsoft SQL, you can often work around this issue by replacing the undefined error message in *vdf.h* with "DT_SZ" (null terminated string). Although this is not the actual datatype, VuGen can compile the script correctly. Please report the problem and send the original script to customer support.

Question 4: What is the meaning of LRD Error 2048?

Answer: VuGen is failing because it is trying to bind a variable with a longer length than what was allocated during recording. You can correct this by enlarging the variable definition in *vdf.h* to receive a longer string back from the database. Search this file for the unique numeric identifier. You will see its definition and length. The length is the third element in the structure. Increase this length as required and the script will replay successfully.

For example, in the following script, we have:

lrd_assign(=;_2_D354, "<ROW_ID>", 0, 0, 0);

In *vdf.h*, we search for *_2_D354* and find

static LRD_VAR_DESC _2_D354 = {

LRD_VAR_DESC_EYECAT, 1, 10, LRD_BYTYPE_ODBC,

{0 ,0, 0}, DT_SZ, 0, 0, 15, 12};

We change it to:

static LRD_VAR_DESC _2_D354 = {

LRD_VAR_DESC_EYECAT, 1, 12, LRD_BYTYPE_ODBC,

{0,, 0}, DT_SZ, 0, 0, 15, 12};

The complete definition of LRD_VAR_DESC appears in *lrd.h*. You can find it by searching for `typedef struct LRD_VAR_DESC`.

Question 5: How can I obtain the number of rows affected by an `UPDATE`, `INSERT` or `DELETE` when using ODBC and Oracle?

Answer: You can use **lrd** functions to obtain this information. For ODBC, use **lrd_row_count**. The syntax is:

int rowcount;

.

.

.

lrd_row_count(Csr33, =;rowcount, 0);

Note that **lrd_row_count** must immediately follow the pertinent statement execution.

For Oracle you can use the fourth argument of **lrd_exec**.

lrd_exec(Csr19, 1, 0, =;rowcount, 0, 0);

If you are using Oracle's OCI 8, you can use the fifth argument of **lrd_ora8_exec**.

lrd_ora8_exec(OraSvc1, OraStm3, 1, 0, =;uliRowsProcessed, 0, 0, 0, 0, 0);

Question 6: How can I avoid duplicate key violations?

Answer: Occasionally, you will see a duplicate key violation when performing an Insert. You should be able to find the primary key by comparing two recordings to determine the problem. Check whether this or earlier `UPDATE` or `INSERT` statement should use correlated queries. You can use the data dictionary in order to find the columns that are used in the violated unique constraint.

In Oracle you will see the following message when a unique constraint is violated:

```
ORA-00001: unique constraint (SCOTT.PK_EMP) violated
```

In this example `SCOTT` is the owner of the related unique index, and `PK_EMP` is the name of this index. Use SQL*Plus to query the data dictionary to find the columns. The pattern for this query is:

```
select column_name from all_ind_columns where index_name =
'<IndexName> and index_owner = '<IndexOwner>';
```

```
select column_name from all_ind_columns where index_name = 'PK_EMP'
and index_owner = 'SCOTT';
```

Since the values inserted into the database are new, they might not appear in earlier queries, but they could be related to the results of earlier queries, such as one more than the value returned in an earlier query.

For Microsoft SQL Server you will see one of these messages:

```
Cannot insert duplicate key row in object 'newtab' with unique index
'IX_newtab'.
Violation of UNIQUE KEY constraint 'IX_Mark_Table'. Cannot insert
duplicate key in object 'Mark_Table'.
Violation of PRIMARY KEY constraint 'PK_NewTab'. Cannot insert
duplicate key in object 'NewTab'.
```

You can use the Query Analyzer to find out which columns used by the key or index. The pattern for this query is:

```
select C.name
    from sysindexes A, sysindexkeys B, syscolumns C
    where C.colid = B.colid and C.id = B.id and
    A.id = B.id and A.indid = B.indid
    and A.name = '<IndexName>' and A.id = object_id('<TableName>')
select C.name
    from sysindexes A, sysindexkeys B, syscolumns C
    where C.colid = B.colid and C.id = B.id and
    A.id = B.id and A.indid = B.indid
    and A.name = 'IX_newtab' and A.id = object_id('newtab')
```

For DB2 you might see the following message:

```
SQL0803N One or more values in the INSERT statement, UPDATE statement,
or foreign key update caused by a DELETE statement are not valid
because they would produce duplicate rows for a table with a primary
key, unique constraint, or unique index. SQLSTATE=23505
```

If you still encounter problems, be sure to check the number of rows changed for Updates and Inserts for both recording and replay. Very often, an `UPDATE` fails to change any rows during replay, because the `WHERE` clause was not satisfied. This does not directly result in an error, but it causes a table not to be properly updated, and can cause a later `SELECT` to select the wrong value when correlating the query.

Also verify that there are no problems during multi-user replay. In certain instances, only one user will successfully perform an `UPDATE`. This occurs with Siebel, where it is necessary to manually write a loop to overcome the problem.

Question 7: The database does not appear to be modified after replaying a script which should have modified the database.

Answer: Through the user application's UI, check if the updated values appear when trying to see the current data accessible to the application. If the values have not been updated, you need to determine they were not changed. Possibly, an `UPDATE` statement changed one or more rows when the application was recorded, and did not change any during replay.

Check these items:

- **Verify statement.** If there is a `WHERE` clause in the `UPDATE` statement, verify that it is correct.
- **Check for correlations.** Record the application twice and compare the `UPDATE` statements from each of the recordings to make sure that the necessary correlations were performed.
- **Check the total number of rows.** Check the number of rows that were changed after the `UPDATE`. For Oracle, this information is stored in the fourth parameter of **lrd_exec**. For ODBC, use **lrd_row_count** to determine the number of rows updated. You can also add code to your script that prints the number of rows that were updated. If this value is 0, the `UPDATE` failed to modify the database.
- **Check the SET clause.** Check the `SET` clause of the `UPDATE` statement. Make sure that you correlated any necessary values here instead of hard-coding them. You can see this by comparing two recordings of the `UPDATE`.

It certain cases, the `UPDATE` works when replaying one Vuser, but not for multiple Vusers. The `UPDATE` of one Vuser might interfere with that of another. Parameterize each Vuser so that each one uses different values during the `UPDATE`, unless you want each vuser to update with the same values. In this case try adding retry logic to perform the `UPDATE` a second time.

Question 8**:** How do I avoid the unique column name error when replaying a statement recorded with an Oracle Application. For example:

lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "

"MTL_UNITS_OF_MEASURE "

"WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "

"SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);

The following error message was issued:

"lrdo.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960: ambiguous column naming in select list".

Answer: Change the statement by adding an alias to at least one of the non-unique columns, thereby mapping it to a new unique name. For example:

lrd_stmt(Csr9,"SELECT UOM_CODE,UOM_CODE second, DESCRIPTION FROM"

"MTL_UNITS_OF_MEASURE "

"WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "

"SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);

Troubleshooting Oracle 2-Tier Vusers

This section contains a list of common problems that you may encounter while working with Oracle Vusers, and suggested solutions.

ORA-20001 and ORA-06512

Errors ORA-20001 and ORA-06512 appear during replay when the lrd_stmt contains the pl/sql block: fnd_signon.audit_responsibility(...)

This statement fails during replay because the sign-on number is unique for each new connection.

Solution

In order to solve this problem you need to use the new correlation tool for the sign-on number. This is second assigned value in the statement.

After you scan for possible values to correlate, highlight the value of the second lrd_assign_bind() for the failed statement. Note that the values in the "correlated query" window may not appear in the same order as the actual recorded statements.

The grid containing the substitution value should appear after the lrd_stmt which contains the pl/sql block: fnd_signon.audit_user(...).

Note: Since the sign-on number is unique for every connection, you need to use correlation for each new connection that you record.

Example of Solution

The following statement failed in replay because the second value, "1498224" is the unique sign-on number for every new connection.

lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"

"; end;", -1, 1, 1, 0);

lrd_assign_bind(Csr6, "s", "D", =;s_D216, 0, 0, 0);

lrd_assign_bind(Csr6, "l", "1498224", =;l_D217, 0, 0, 0);

lrd_assign_bind(Csr6, "f", "1", =;f_D218, 0, 0, 0);

lrd_assign_bind(Csr6, "a", "810", =;a_D219, 0, 0, 0);

lrd_assign_bind(Csr6, "r", "20675", =;r_D220, 0, 0, 0);

lrd_assign_bind(Csr6, "t", "Windows PC", =;t_D221, 0, 0, 0);

lrd_assign_bind(Csr6, "p", "", =;p_D222, 0, 0, 0);

lrd_exec(Csr6, 1, 0, 0, 0, 0);

The sign-on number can be found in the lrd_stmt with "fnd_signon.audit_user". The value of the first placeholder "a" should be saved. The input of "a" is always "0" but the output is the requested value.

Modified code:

lrd_stmt(Csr4, "begin fnd_signon.audit_user(:a,:l,:u,:t,:n,:p,:s); end;", -1, 1, 1, 0);

lrd_assign_bind(Csr4, "a", "0", =;a_D46, 0, 0, 0);

lrd_assign_bind(Csr4, "l", "D", =;l_D47, 0, 0, 0);

lrd_assign_bind(Csr4, "u", "1001", =;u_D48, 0, 0, 0);

lrd_assign_bind(Csr4, "t", "Windows PC", =;t_D49, 0, 0, 0);

lrd_assign_bind(Csr4, "n", "OraUser", =;n_D50, 0, 0, 0);

```
lrd_assign_bind(Csr4, "p", "", =;p_D51, 0, 0, 0);

lrd_assign_bind(Csr4, "s", "14157", =;s_D52, 0, 0, 0);

lrd_exec(Csr4, 1, 0, 0, 0, 0);

lrd_save_value(=;a_D46, 0, 0, " saved_a_D46");

Grid0(17);

lrd_stmt(Csr6, "begin fnd_signon.audit_responsibility(:s,:l,:f,:a,:r,:t,:p)"

"; end;", -1, 1, 1, 0);

lrd_assign_bind(Csr6, "s", "D", =;s_D216, 0, 0, 0);

lrd_assign_bind(Csr6, "l", " <saved_a_D46>", =;l_D217, 0, 0, 0);

lrd_assign_bind(Csr6, "f", "1", =;f_D218, 0, 0, 0);

lrd_assign_bind(Csr6, "a", "810", =;a_D219, 0, 0, 0);

lrd_assign_bind(Csr6, "r", "20675", =;r_D220, 0, 0, 0);

lrd_assign_bind(Csr6, "t", "Windows PC", =;t_D221, 0, 0, 0);

lrd_assign_bind(Csr6, "p", "", =;p_D222, 0, 0, 0);

lrd_exec(Csr6, 1, 0, 0, 0, 0);
```

Working with large numbers

Large numbers (NUMBER data type) sometimes appear in different format in the GRID and in the ASCII file. This difference makes it more difficult to identify numbers while searching for values to save for correlation.

For example, you could have a value appear as 1000003 in the grid, but as 1e+0006 in the Recording Log (ASCII file).

Workaround

If you have an error during replay and the correlation tool cannot locate the value in previous results, look for this value in the other format in grid.

ORA-00960

This error may occur with non-unique column names. For example:

```
lrd_stmt(Csr9, "SELECT UOM_CODE, UOM_CODE, DESCRIPTION FROM "

"MTL_UNITS_OF_MEASURE "

"WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "

"SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);
```

In this case you receive the following error:

"lrdo.c/fjParse: "oparse" ERROR return-code=960, oerhms=ORA-00960: ambiguous column naming in select list".

Workaround

Change the statement by adding an alias to at least one of the non-unique columns, thus mapping it to a new unique name. For example:

lrd_stmt(Csr9,"SELECT UOM_CODE,UOM_CODE second, DESCRIPTION FROM"

"MTL_UNITS_OF_MEASURE "

"WHERE NVL(DISABLE_DATE, SYSDATE + 1) > "

"SYSDATE ORDER BY UOM_CODE", -1, 1, 1, 0);

Alternate Workaround: remove ORDER BY from the lrd statement.

ORA-2002

Error 2002 appears when you try to use an unopened cursor. It occurs when you replay a user more than one iteration and you recorded into more than one section of the script.

Specifically, if a cursor is opened in the vuser_init section and closed in the Actions section, then you will encounter this error on the second iteration if you try to use the cursor. This is because it was closed but not re-opened.

For example: You have *lrd_open_cursor* in the vuser_init section and *lrd_close_cursor* in the Actions section. If you replay this user more than one iteration, you are going to get an error in the second iteration because you try using an unopened cursor (it was closed in first iteration, but not re-opened in the second).

Workaround

The easiest way to solve this is to move the **lrd_close_cursor** or/and **lrd_close_connection** of the problem cursor to the *vuser_end* section.

Database Protocols (lrd)

Replay of recorded asynchronous operations is not supported.

Wrong Client Version

You may receive an error message when running the wrong Oracle client version:

"Error: lrdo_open_connection: "olog" LDA/CDA return-code_019: unable to allocate memory in the user side"

Workaround

You need to modify the library information in the *lrd.ini* file, located in the your product's bin folder. This file contains the settings that indicate which version of database support is loaded during recording or replay. The file contains a section for each type of host. For example, the following section of the *lrd.ini* file is for Oracle on HP/UX:

[ORACLE_HPUX]

;816=liblrdhpo816.sl
;81=liblrdhpo81.sl
;80=liblrdhpo80.sl
73=liblrdhpo73.sl
72=liblrdhpo72.sl

These settings indicate that Vusers should use the LoadRunner library liblrdhpo816.sl if the client uses Oracle 8.1.6, liblrdhpo81.sl for Oracle 8.1.5, and so on.

During replay on Linux, the settings in the lrd *ini* file must indicate the correct version of the database to use. Suppose it is necessary to replay a Vuser for HP/UX using Oracle 8.1.5. In that case the previous lines for other versions of Oracle should be commented out with a ";" at the beginning of the line.

This section of the lrd.ini file will now look like:

[ORACLE_HPUX]

;816=liblrdhpo816.sl
81=liblrdhpo81.sl
;80=liblrdhpo80.sl
73=liblrdhpo73.sl
72=liblrdhpo72.sl

You also may need to make a change for Win32 if the application does not use the DLL mentioned in the lrd.ini file. For example, PowerBuilder 6.5 uses Oracle 8.0.5, but it uses the ora803.dll, not the ora805.dll. In that case, either comment out the 805 and 804 sections of the ORACLE_WINNT section, or change the 805 section from:

805=lrdo32.dll+ora805.dll

to

805=lrdo32.dll+ora803.dll

Enterprise Java Beans (EJB) Protocol

EJB Testing Overview

VuGen provides several tools for developing a script that tests Java applications. For generating a Vuser script through recording, use the Java Record and Replay Vuser. For creating a script through programming, use the Custom Java Vuser type.

EJB Testing Vusers differ from the standard Java Vusers in that VuGen automatically creates a script to test or tune EJB functionality without recording or programming. Before you generate a script, you specify the JNDI properties and other information about your application server. VuGen's EJB Detector scans the application server and determines which EJBs are available. You select the EJB that you want to test or tune, and VuGen generates a script that emulates each of the EJB's methods.

It creates transactions for each method so that you can measure its performance and locate problems. In addition, each method is wrapped in a **try and catch** block for exception handling.

Note that in order to create EJB testing scripts, the EJB Detector must be installed and active on the application server host. The Detector is described in the following sections.

VuGen also has a built-in utility for inserting methods into your script. Using this utility, you display all of the available packages, select the desired methods, and insert them into your script. For more information, see "How to Run an EJB Vuser Script" on page 503.

EJB Detector Output and Log Files

You can examine the output of the EJB Detector to see if it has detected all the active EJBs. The output log shows the paths being checked for EJBs. At the end of the scan, it displays a list of the EJBs that were found, their names and locations. For example:

```
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_beanManaged.jar...
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_
statefulSession.jar...
Checking EJB Entry: f:/weblogic/myserver/ejb_basic_
statelessSession.jar...
------------------------ Found 3 EJBs --------------------------
** PATH: f:/weblogic/myserver/ejb_basic_beanManaged.jar
- BEAN: examples.ejb.basic.beanManaged.AccountBean
** PATH: f:/weblogic/myserver/ejb_basic_statefulSession.jar
- BEAN: examples.ejb.basic.statefulSession.TraderBean
** PATH: f:/weblogic/myserver/ejb_basic_statelessSession.jar
- BEAN: examples.ejb.basic.statelessSession.TraderBean
```

If no EJBs were detected (that is, "Found 0 EJBs"), check that the EJB jar files are listed in the "`Checking EJB Entry:...`" lines. If they are not listed, check that the **search root dir** path is correct. If they are being inspected but still no EJBs are detected, check that these EJB jar files are deployable (can be successfully deployed into an application server). A deployable jar file should contain the Home Interface, Remote Interface, Bean implementation, the Deployment Descriptor files (xml files, or .ser files), and additional vendor-specific files.

If you still encounter problems, set the debug properties in the **detector.properties** file, located in the DETECTOR_HOME\classes directory, to retrieve additional debug information.

After the EJBs are detected, the HTTP Server is initialized and waits for requests from the VuGen EJB-Testing Vuser. If there are problems in this communication process, enable the property **webserver.enableLog** in the **webserver.properties** file located in the DETECTOR_HOME\classes directory.

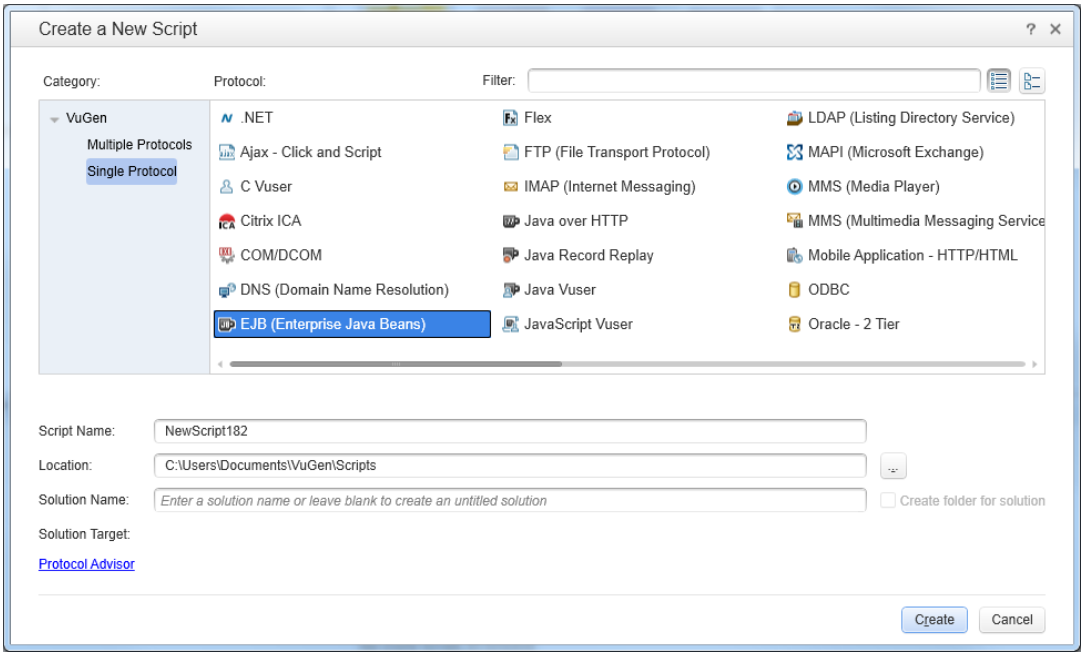
This enables printouts of additional debug information, and other potentially important error messages in the **webserver.log** file.

How to Create an EJB Vuser Script

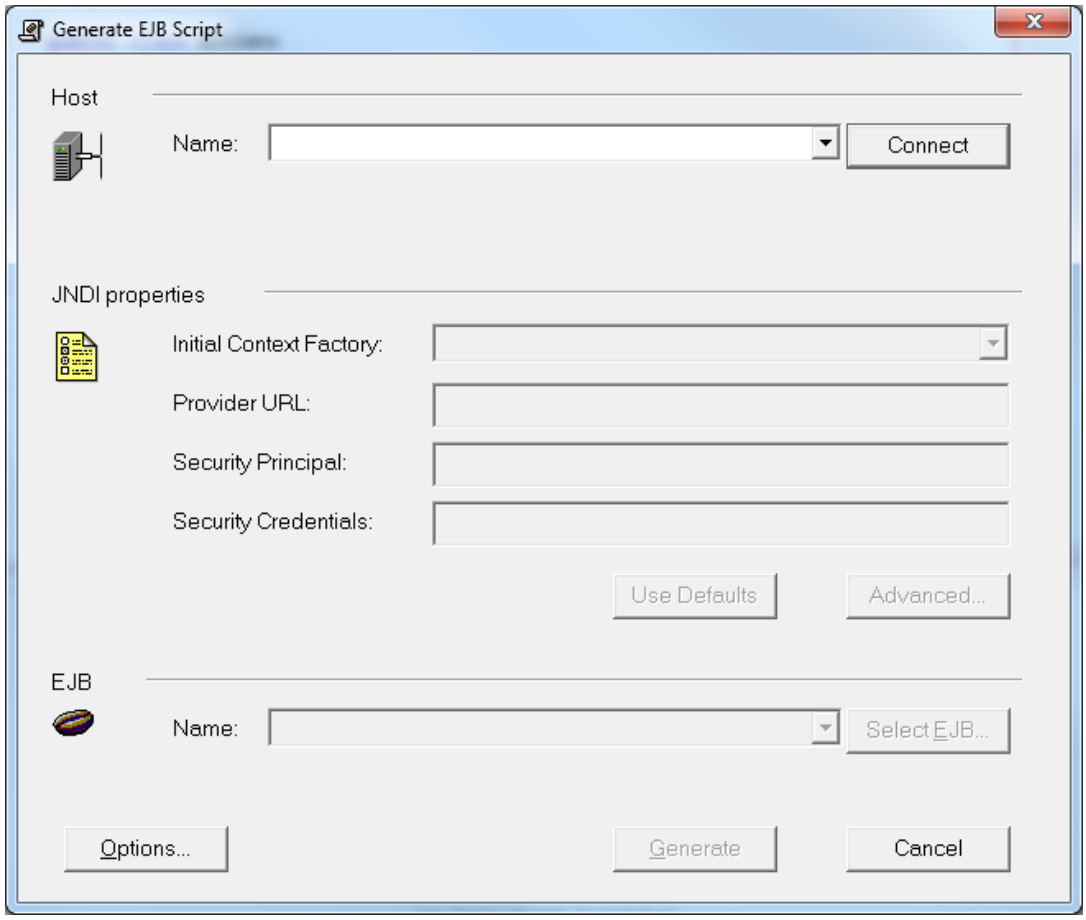
Creating an EJB Vuser Script differs from creating Vuser Scripts using other protocols. This task describes how to create an EJB Vuser Script.

1. Select **File > New Script and Solution** or click the **New script** button. The Create a New

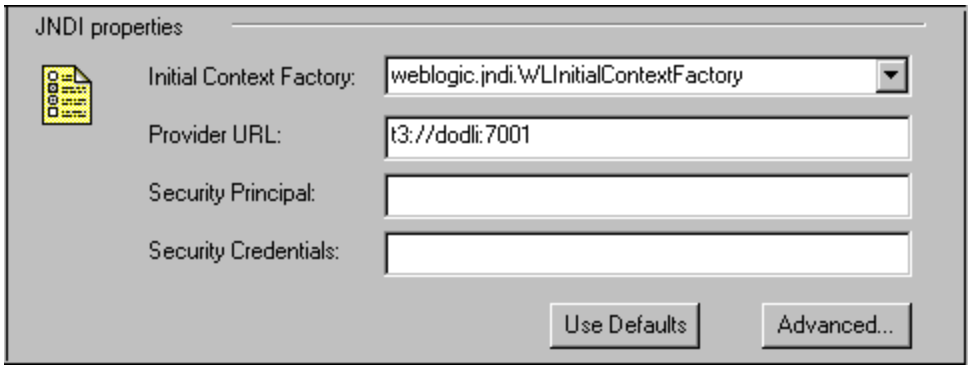
Script dialog box opens.



2. Select **EJB (Enterprise Java Beans)** from the **Protocol** list and then click **Create**. VuGen opens a blank EJB Vuser script.
3. Click **Record > Record**. The Generate EJB Script dialog box opens.



4. Specify a machine on which VuGen's EJB Detector is installed. Note that the Detector must be running in order to connect. Click **Connect**. The **JNDI properties** section is enabled.



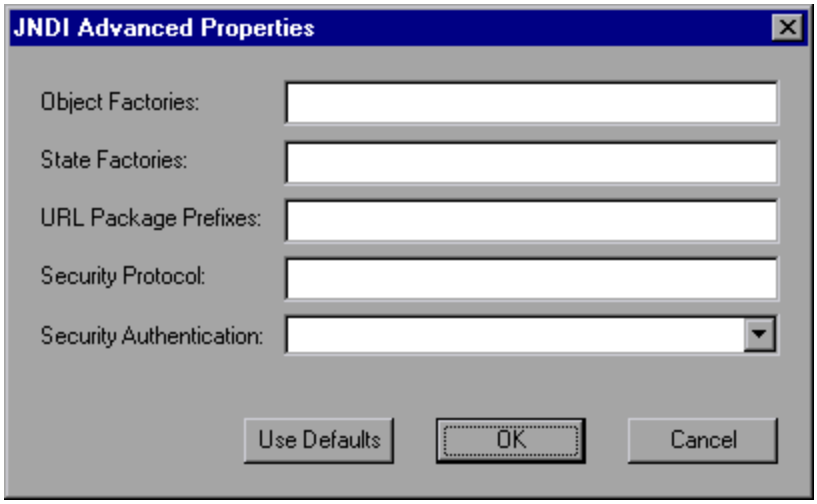
5. The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory and** the **Provider URL**.

   If your application server requires authentication, enter the user name in the **Security Principal** box and a password in the **Security Credentials** box.

   Here are the default values of the two JNDI mandatory properties:

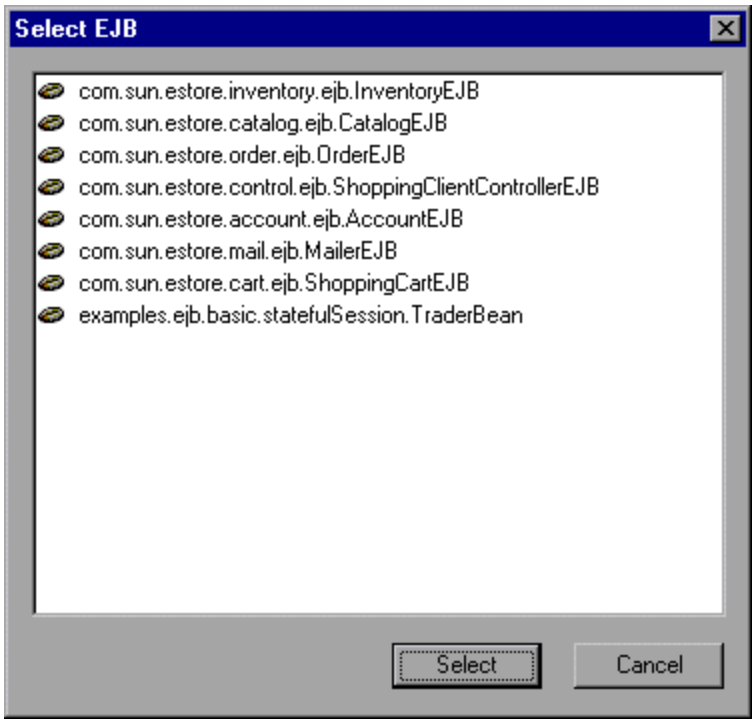
Type	Initial Context Factory	Provider URL
WebLogic	weblogic.jndi. WLInitialContextFactory	t3://<appserver_host>:7001
WebSphere 3.x	com.ibm.ejs.ns.jndi. CNInitialContextFactory	iiop://<appserver_host>:900
WebSphere 4.x	com.ibm.websphere.naming. WsnInitialContextFactory	iiop://<appserver_host>:900
Sun J2EE	com.sun.enterprise.naming. SerialInitContextFactory	N/A
Oracle	com.evermind.server. AppplicationClient InitialContextFactory	ormi://<appserver_host>/ <application_name> (the app. name of the EJB in <oc4j>/config/server.xml)

6. To set advanced properties for the JNDI, click **Advanced** to open the JNDI Advanced Properties dialog box.



Specify the desired properties: **Object Factory**, **State Factory**, **URL Package Prefixes**, **Security Protocol**, and **Security Authentication**. Click **OK**.

7. In the **EJB** section of the dialog box, click **Select** to select the EJB for which you want to create a test. A dialog box opens with a list of all the EJBs currently available to you from the application server.



8. Highlight the EJB you want to test and click **Select**.
9. In the Generate EJB Script dialog box, click **Generate**. VuGen creates a script with Java Vuser functions. The script contains code that connects to the application server and executes the EJB's methods.
10. Save the script.

    Note that you cannot generate test code for an additional EJB, within an existing script. To create a test for another EJB, open a new script and repeat steps 2-9.

How to Run an EJB Vuser Script

After you generate a script for your EJB testing, and make the necessary modifications, you are ready to run your script. The EJB script allows you to perform two types of testing: functional and load. The functional testing verifies that the EJB, functions properly within your environment. The load testing allows you to evaluate the performance of the EJB when executing many users at one time.

Run a functional test

1. Modify the default values that were automatically generated.
2. Insert value checks using the **lr.value_check** method. These methods were generated as comments in the script (see "Invoking the EJB Methods" on page 510).
3. Insert additional methods, and modify their default values. For more information, see the section on inserting Java functions in "Java Protocol" on page 529.
4. Configure the Run-Time Settings. For more information, see "Run-Time Settings" on page 332.
5. Make sure you have a valid Java environment. For more information, see "How to Set and

Verify the Java Environment" on page 506.

6. Run the script. Click the **Run** button or select **Replay > Run**. View the Execution Log node to view any run-time errors. The execution log is stored in the **mdrv.log** file in the script's folder. A Java compiler (Sun's javac), checks it for errors and compiles the script.

How to Install and Run the EJB Detector

This task describes how to install and run the EJB Detector. The EJB Detector is a separate agent that must be installed on each machine that is being scanned for EJBs. This agent detects the EJBs on the machine. Before installing the EJB Detector, verify that you have a valid JDK environment on the machine.

1. **Install the EJB Detector**
   a. Verify that you have a valid JDK environment on the machine.
   b. Create a home directory for the EJB Detector on the application server machine, or on the client machine (and mount the file systems as mentioned).
   c. Unzip the **<LR_root>\ejbcomponent\ejbdetector.jar** file into the EJB Detector directory.
2. **Run the EJB Detector**

   The EJB Detector must be running before you start the EJB script generation process in VuGen. You can either run the EJB detector on the application server or on the client machine (in this case, make sure to mount to the application server from the EJB Detector (client) machine, specify the mount directory in the search root directory, and change the generated script to connect to the mounted machine, instead of the local machine).

   - To run the EJB Detector from the command line, see "How to Run the EJB Detector from the command line" below.
   - To run the EJB Detector from a batch file, see "How to Run the EJB Detector from a batch file" on next page.

How to Run the EJB Detector from the command line

This task describes how to run the EJB Detector from the command line

1. Add the DETECTOR_HOME\classes and the DETECTOR_HOME\classes\xerces.jar to the CLASSPATH environment variable.
2. If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested as well as the following vendor EJB classes to the CLASSPATH:
3. For WebLogic 4.x: `<WebLogic directory>\lib\weblogicaux.jar`
4. For WebSphere 3.x: `<WebSphere directory>\lib\ujc.jar`
5. If your EJBs use additional classes directory or .jar files, add them to the CLASSPATH.
6. Enter the following string in the command line:

```
java EJBDetector [search root dir] [listen port]
```

search root dir	One or more directories or files in which to search for EJBs (separated by semicolons). Follow these guidelines: BEA WebLogic Servers 4.x and 5.x. Specify the application server root directory. BEA WebLogic Servers 6.x. Specify full path of the domain folder. WebSphere Servers 3.x. Specify the full path of the deployed EJBs folder. WebSphere Servers 4.0. Specify the application server root directory. Oracle OC4J. Specify the application server root directory. Sun J2EE Server. Specify the full path to the deployable .ear file or directory containing a number of .ear files. If unspecified, the classpath will be searched.
listen port	The listening port of the EJB Detector. The default port is 2001. If you change this port number, you must also specify it in the Host name box of the Generate EJB Test dialog box. For example, if your host is metal, if you are using the default port, you can specify metal. If you are using a different port, for example, port 2002, enter metal:2002.

How to Run the EJB Detector from a batch file

You can launch the EJB detector using a batch file, **EJB_Detector.cmd**. This file resides in the root folder of the EJB Detector installation, after you unzip **ejbdetector.jar**.

Run the EJB Detector from a batch file

1. Open **env.cmd** in the EJB Detector root folder, and modify the following variables according to your environment:

JAVA_HOME	The root folder of JDK installation
DETECTOR_INS_DIR	The root folder of the Detector installation
APP_SERVER_DRIVE	The drive hosting the application server installation
APP_SERVER_ROOT	Follow these guidelines: BEA WebLogic Servers 4.x and 5.x. Specify the application server root folder. BEA WebLogic Servers 6.x. Specify full path of the domain folder. WebSphere Servers 3.x. Specify the full path of the deployed EJBs folder. WebSphere Servers 4.0. Specify the application server root folder. Oracle OC4J. Specify the application server root folder. Sun J2EE Server. Specify the full path to the deployable .ear file or folder containing a number of .ear files.
EJB_DIR_LIST (optional)	List of directories/files, separated by `;' and containing deployable .ear/.jar files, and any additional classes folder or .jar files or used by your EJBs under test.

2. Save **env.cmd**.
3. If you are working with EJB1.0 (Weblogic 4.x, WebSphere 3.x), add the classes of EJBs that are being tested, as well as the following vendor EJB classes, to the CLASSPATH in the env file:
   - For WebLogic 4.x: `<WebLogic directory>\lib\weblogicaux.jar`
   - For WebSphere 3.x: `<WebSphere directory>\lib\ujc.jar`
4. Run the **EJB_Detector.cmd** or **EJB_Detector.sh** (Linux platforms) batch file to collect information about the deployable applications containing EJBs, for example:

   ```
   C:\>EJB_Detector [listen_port]
   ```

   where `listen_port` is an optional argument specifying a port number on which the EJB Detector will listen for incoming requests (default is 2001).

How to Set and Verify the Java Environment

This task describes the methods of setting and verifying the java environment.

For Websphere 3.x users:

1. Using the IBM JDK 1.2 or higher:
   - Add the `<WS>\lib\ujc.jar to the` classpath.
2. Using the Sun JDK 1.2.x:
   - Remove the file <JDK>\jre\lib\ext\iiimp.jar
   - Copy the following files from the <WS>\jdk\jre\lib\ext folder to the <JDK>\jre\lib\ext directory: iioprt.jar, rmiorb.jar.
   - Copy the 'ujc.jar' from the <WS>\lib folder, to <JDK>\jre\lib\ext.
   - Copy the file <WS>\jdk\jre\bin\ioser12.dll to the <JDK>\jre\bin folder.

   where <WS> is the home folder of the WebSphere installation and <JDK> is the home folder of the JDK installation.

   Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

For WebSphere 4.0 users:

Make sure that you have valid Java environment on your machine of IBM JDK1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<WS>/lib/webshpere.jar;
<WS>/lib/j2ee.jar;
```

Where <WS> is the home folder of the WebSphere installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

Note: If your application server is installed on a Linux machine or if you are using WebSphere

3.0.x, you must install IBM JDK 1.2.x on the client machine to obtain the required files.

For Oracle OC4J users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher (JDK1.3 preferable). Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<OC4J>/orion.jar;<OC4J>/ejb.jar;<OC4J>/jndi.jar; ;<OC4J>/xalan.jar;
<OC4J>/crimson.jar
```

where <OC4J> is the home folder of the application server installation.

For Sun J2EE users:

Make sure that you have valid Java environment on your machine of JDK1.2 or higher. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entries to the Additional Classpath section:

```
<J2EE>/j2ee.jar;<AppClientJar>
```

where <J2EE> is the home folder of the application server installation and <AppClientJar> is the full path to the application client jar file created automatically by the sdk tools during the deployment process.

For WebLogic 4.x - 5.x Users:

Make sure that you have valid Java environment on your machine (path =; classpath). Open the Run-Time Settings dialog box and select the Java VM node. Add the following two entries to the Additional Classpath section:

```
<WL>/classes;<WL>/lib/weblogicaux.jar
```

where <WL> is the home folder of the WebLogic installation.

For WebLogic 6.x and 7.0 users:

Make sure that you have valid Java environment on your machine (path =; classpath). WebLogic 6.1 requires JDK 1.3. Open the Run-Time Settings dialog box and select the Java VM node. Add the following entry to the Additional Classpath section:

```
<WL>/lib/weblogic.jar;    // Weblogic 6.x
<WL>/server/lib/weblogic.jar    // Weblogic 7.x
```

where <WL> is the home folder of the WebLogic installation.

Clear the **Use -Xbootclasspath VM parameter** check box to turn off this option.

Generate EJB Script Dialog Box

Enables you to begin recording an EJB script.

To Access	**File > New Script > EJB (Enterprise Java Beans)**
Relevant Tasks	"How to Install and Run the EJB Detector" on page 504

User interface elements are described below:

UI Element	Description
Advanced...	Opens the JNDI Advanced Properties dialog box. Specify the desired properties.
Options...	Opens relevant recording options dialog box. For more information see "Recording Options" on page 258.
Connect Disconnect	Connects to and disconnects from the server. You must be connect to the server to generate a script.
Generate	Generates the Vuser script.
Select EJB...	Opens the Select EJB dialog box.
EJB Name	The name of EJB for which you want to create a test.
Host Name	The machine on which the EJB Detector is installed. The detector must be running in order to connect.
JNDI Properties	The EJB Detector automatically detects the default JNDI properties. You can manually modify these properties in the appropriate edit boxes. The properties you can modify are a string for the **Initial Context Factory** and the **Provider URL**.

EJB Vuser Script Examples

VuGen generates a script that tests your EJB, based on the JNDI (Java Naming and Directory Interface) properties you specified when creating the Vuser script. JNDI is Sun's programming interface used for connecting Java programs to naming and folder services such as DNS and LDAP.

Each EJB Vuser script contains three primary parts:

- Locating the EJB Home Using JNDI
- Creating an Instance
- Invoking the EJB Methods

Locating the EJB Home Using JNDI

The first section of the script contains the code that retrieves the JNDI properties. Using the specified context factory and provider URL, it connects to the application server, looks up the specified EJB and locates the EJB Home.

In the following example, the JNDI Context Factory is weblogic.jndi.WLInitialContextFactory, the URL of the provider is t3://dod:7001 and the JNDI name of the selected EJB is carmel.CarmelHome.

```
public class Actions
{
    public int init() {
        CarmelHome _carmelhome = null;
        try {
            // get the JNDI Initial Context
            java.util.Properties p = new java.util.Properties();
            p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
            p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001");
            javax.naming.InitialContext _context = new
javax.naming.InitialContext(p);
            // lookup Home Interface in the JNDI context and narrow it
            Object homeobj = _context.lookup("carmel.CarmelHome");
            _carmelhome = (CarmelHome)
javax.rmi.PortableRemoteObject.narrow(homeobj, CarmelHome.class);
        } catch (javax.naming.NamingException e) {
            e.printStackTrace();
        }
```

If the script is generated with an EJB Detector running on the client rather than an application server, you must manually modify the URL of the provider. For example, in the following line, the provider specifies dod as the EJB detector host name:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://dod:7001")
```

Replace the recorded host name with the application server name, for example:

```
p.put(javax.naming.Context.PROVIDER_URL, "t3://bealogic:7001")
```

You can specify the provider URL before recording, so you don't have to modify it manually, in the **JDNI Properties** section of the Generate EJB Script dialog.

Creating an Instance

Before executing the EJB methods, the script creates a Bean instance for the EJB. The creation of the instance is marked as a transaction to allow it to be analyzed after the script is executed. In addition, the process of creating an instance is wrapped in a **try and catch** block providing exception handling.

For Session Beans. Use the EJB home 'create' method to get a new EJB instance.

In the following example, the script creates an instance for the Carmel EJB.

```
        // create Bean instance
        Carmel _carmel = null;
        try {
            lr.start_transaction("create");
            _carmel = _carmelhome.create();
            lr.end_transaction("create", lr.AUTO);
        } catch (Throwable t) {
            lr.end_transaction("create", lr.FAIL);
```

```
            t.printStackTrace();
        }
```

For Entity Beans - use the **findByPrimaryKey** method to locate the EJB instance in an existing database, and if not found, then use the **create** method, to create it there.

In the following example, the script attempts to locate an instance for the account EJB, and if it fails then creates it.

```
// find Bean instance
try {
    com.ibm.ejs.doc.account.AccountKey _accountkey = new
com.ibm.ejs.doc.account.AccountKey();
    _accountkey.accountId = (long)0;
    lr.start_transaction("findByPrimaryKey");
    _account = _accounthome.findByPrimaryKey(_accountkey);
    lr.end_transaction("findByPrimaryKey", lr.AUTO);
} catch (Throwable thr) {
    lr.end_transaction("findByPrimaryKey", lr.FAIL);
    lr.message("Couldn't locate the EJB object using a primary key.
Attempting to manually create the object... ["+thr+"]");
    // create Bean instance
    try {
        lr.start_transaction("create");
        _account = _accounthome.create
((com.ibm.ejs.doc.account.AccountKey)null);
        lr.end_transaction("create", lr.AUTO);
    } catch (Throwable t) {
        lr.end_transaction("create", lr.FAIL);
        t.printStackTrace();
    }
}
```

You can use other find methods, supplied by your Entity Bean, to locate the EJB instance. For example:

```
// get an enumeration list of all Email EJB instances that represents
// the name 'John' in the database.
Enumeration enum = home.findByName("John");
    while (enum.hasMoreElements()) {
                Email addr = (Email)enum.nextElement();
        ...
}
```

Invoking the EJB Methods

The final part of the script contains the actual methods of the EJB. Each method is marked as a transaction to allow it to be analyzed after running the script. In addition, each method is wrapped in a try and catch block providing exception handling. When there is an exception, the transaction is marked as failed, and the script continues with the next method. VuGen creates a separate block for each of the EJB methods.

```
// ------- Methods ------------
        int _int1 = 0;
        try {
            lr.start_transaction("buyTomatoes");
            _int1 = _carmel.buyTomatoes((int)0);
            //lr.value_check(_int1, 0, lr.EQUALS);
            lr.end_transaction("buyTomatoes", lr.AUTO);
        } catch (Throwable t) {
            lr.end_transaction("buyTomatoes", lr.FAIL);
            t.printStackTrace();
        }
```

VuGen inserts default values for the methods, for example, 0 for an integer, empty strings ("") for Strings, and NULL for complex Java objects. If necessary, modify the default values within the generated script.

```
            _int1 = _carmel.buyTomatoes((int)0);
```

The following example shows how to change the default value of a non-primitive type using parameterization:

```
Detail details = new Details(<city>,<street>,<zip>,<phone>);
JobProfile job = new JobProfile(<department>,<position>,<job_type>);
Employee employee=new Employee(<first>,<last>, details, job,
<salary>);
_int1 = _empbook.addEmployee((Employee)employee);
```

For methods that return a primitive, non-complex value or string, VuGen inserts a commented method lr.value_check. This method allows you to specify an expected value for the EJB method. To use this verification method, remove the comment marks (//) and specify the expected value. For example, the carmel.buyTomatoes method returns an integer.

```
            _int1 = _carmel.buyTomatoes((int)0);
            //lr.value_check(_int1, 0, lr.EQUALS);
```

If you expect the method to return a value of 500, modify the code as follows:

```
            _int1 = _carmel.buyTomatoes((int)0);
            lr.value_check(_int1, 500, lr.EQUALS);
```

If you want to check if the method does not return a certain value, modify the code as follows:

```
            _int1 = _carmel.buyTomatoes((int)0);
            lr.value_check(_int1, 10, lr.NOT_EQUALS);
```

If the expected value is not detected, an exception occurs and the information is logged in the output window.

```
System.err: java.lang.Exception: lr.value_check failed.[Expected:500
Actual:5000]
```

EJB Vuser scripts support all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes "//".

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause

inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. (see Run-Time settings) This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

Flex (RTMP/AMF) Protocol

Flex Overview

Flex is a collection of technologies that provides developers with a framework for building RIAs (Rich Internet Applications) based on the Flash Player.

RIAs are lightweight online programs that provide users with more dynamic control than with a standard Web page. Like Web applications built with Ajax, Flex applications are more responsive, because the application does not need to load a new Web page every time the user performs an action. However, unlike working with Ajax, Flex is independent of browser implementations such as JavaScript or CSS. The framework runs on Adobe's cross-platform Flash Player.

Flex applications consist of many MXML and ActionScript files. They are compiled into a single SWF movie file which can be played by the Flash player installed on the client's browser.

Flex supports a variety of client/server communication methods, such as RPC, Data Management, and Real-Time messaging. It supports several data formats such as HTTP, AMF, and SOAP.

VuGen lets you create a Vuser script that emulates communication with Flex 2 RPC services. VuGen's **Flex** protocol lets you create scripts that emulate Flex applications working with AMF0 and AMF3 or HTTP data. For Flex applications working with SOAP data, use the **Web Services** Vuser protocol.

Flex Correlations

Support for correlations has been added for the following steps:

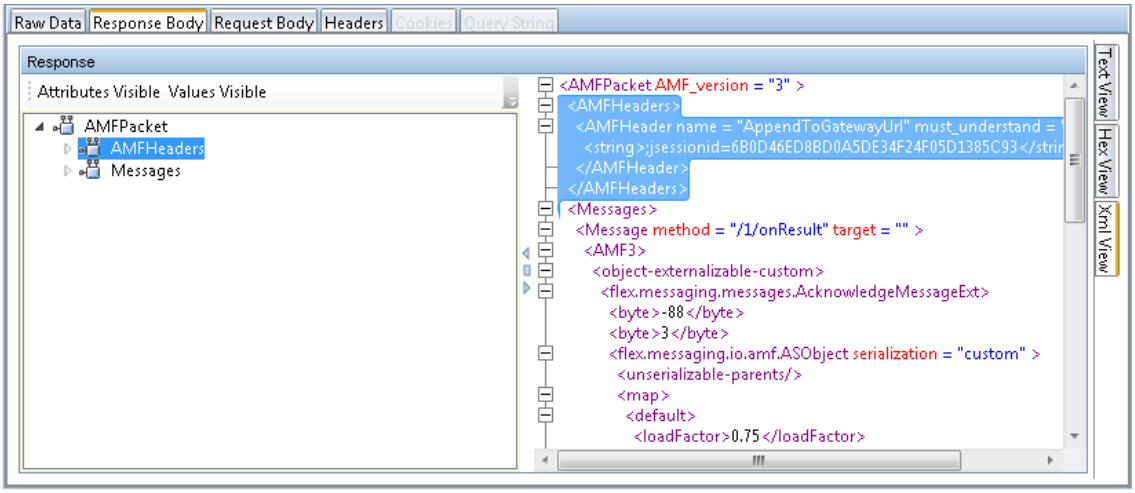
- flex_login
- flex_logout
- flex_ping
- flex_amf_call
- flex_remoting_call
- flex_rtmp_tunnelled_connect
- flex_rtmp_tunnelled_send

In addition, a new snapshot has been designed to show data in several views:

- Raw Data
- Response Body
- Request Body
- Header Body

If the Response Body and the Request Body are in XML format, the data can be displayed as:

- Text
- Hex
- XML



Support has been added for the following features:

- **Correlations rules**

  DSid, **jsessionid**, and **RTMPT ID** have been added.
- **Design Studio**
- **Manual correlation** using the API **web_reg_save_param_xpath**.

For details, see "How to Correlate Scripts Using Design Studio" on page 137.

Code Generation Notification

If a Flex, Silverlight, or Java over HTTP script encounters an error during the code generation phase, VuGen issues a warning. This warning appears in the Errors pane, when the **Warnings** button is selected, and the **Define Available Categories** filter is set to **All** or **Code Generation Notification**.The list of warnings displays details about each error, as well as recommended actions for resolving the problem. Follow the recommended actions and regenerate the script.

To manually open the Errors pane at any time, select **View > Errors**.

Parsing Responses in Flex Scripts

When generating a Flex script, LoadRunner attempts to parse responses for any of the following steps:

• flex_amf_call

• flex_remoting_call

• flex_login

• flex_logout

• flex_ping

If the parsing fails, the following attribute is dynamically added to the step:

IsParseResponse = No

This instructs LoadRunner not to parse the responses for that step during script replay. Every time you regenerate the script, LoadRunner will attempt to parse again, and will set this parameter to false if it fails. If needed, you can delete this line, or set the value to = 'Yes' to force LoadRunner to parse responses for that step during replay.

Additionally, you can manually add the attribute and set the value to 'No' in a generated script, even if the parse is successful, as it may enhance replay performance.

RTMP Tunneled

- **Messaging support**. The Flex protocol supports enhanced record and replay of messaging and has been verified for Adobe LiveCycle Data Services ES2 Version 3.1.
- **Streaming support**. The Flex protocol supports enhanced record and replay of streaming. For details, see "RTMP/RTMPT Streaming" on next page.

  When you record a Flex stream, by default, LoadRunner generates a single **flex_rtmp_tunneled_send** step in place of many **flex_rtmp_tunneled_send** steps. The flex_rtmp_tunneled_send step makes your script more readable (eliminating tens or hundreds of lines) and makes the replay more reliable.

  > **Note:** The new **flex_rtmp_tunneled_send** step is generated when the **Generate single RTMP/T step** option is selected in the **Flex:RTMP** pane of the **Recording Options** dialog box. Although this step makes the script more reliable, it does not replay certain actions you may perform while recording your script, such as pause and seek. It also does not replay the automatic requests that the client performs when Dynamic Stream is in use.

  If it is important to replay these actions, you can clear the **Generate single RTMP/T step** option in the **Flex> RTMP** pane of the **Recording Options** dialog box, which causes LoadRunner to generate the steps for all of the raw streaming data.

  However, to ensure proper replay, you must manually modify the generated script as described in "RTMP/RTMPT Streaming" on next page.

  The above functionality has been verified for Flash Media Server versions 3.5 and 4.
- **Externalizable objects**. LoadRunner supports externalizable objects over RTMP Tunneled.
- **User Data Points**. LoadRunner generates a number of new data points that provide more useful information for analysis.
- The Flex RTMP-Tunneled protocol supports manual correlation using **web_reg_save_param_xpath API**.

For additional details on **flex_rtmp_tunneled_send** including a complete example, see the Function Reference (**Help > Function Reference**).

RTMP/RTMPT Streaming

LoadRunner's Flex protocol supports recording and replay of streaming for both the RTMP and RTMPT protocols. You can record using either the regular recording mode or the simplified recording mode. The simplified mode enables VuGen to generate a single function in place of the multiple functions that are generated when the regular recording mode is used. When you use the simplified mode to record, the following occurs:

- For an RTMP-based stream: VuGen generates a single **flex_rtmp_receive_stream** step in place of many **flex_rtmp_receive** and **flex_rtmp_send** steps.
- For an RTMPT-based stream: VuGen generates a single modified **flex_rtmp_tunneled_send** step in place of many **flex_rtmp_tunneled_send** steps.

The single generated **flex_rtmp_receive_stream** or **flex_rtmp_tunneled_send** step makes the Vuser script more readable (by eliminating multiple lines of code), and makes the script replay more reliable. It is recommended that you use the simplified mode for recording your Vuser scripts, unless the Vuser activity includes asynchronous behavior, as described below.

The simplified recording mode is the default recording mode for streaming. To activate the simplified mode, open the Recording Options dialog box, click **Flex > RTMP**, and select the **Generate single step for RTMP/T stream handling** check box.

The differences between the simplified and regular recording modes are listed below:

	Simplified mode	Regular mode
(Recording option) Generate single step for RTMP/T stream handling check box	Selected	Not selected
Functions generated	RTMP: Generates **flex_rtmp_receive_stream** functions. RTMPT: Generates **flex_rtmp_tunneled_send** functions.	Generates **flex_rtmp_receive** and **flex_rtmp_send** steps.
Number of functions generated per stream	One	Multiple
Supports asynchronous behavior	No	Yes
Default Mode	Yes	No

Note: The simplified recording mode is supported for Flash Media Server versions 3.5 and 4.

Synchronous Vuser behavior

The **flex_rtmp_receive_stream** and **flex_rtmp_tunneled_send** functions that are generated when the simplified recording mode is selected are synchronous functions. This means that no other Vuser functions can be executed while either of these functions is executing. For example,

consider a Vuser script that includes a **flex_rtmp_receive_stream** function that streams a video for 5 minutes. During the 5 minute period during which the video is streaming, the Vuser will not be able to perform any other actions, such as clicking the **Pause** button or skipping to a different location in the video. Clicking a button while a video is streaming is an example of asynchronous behavior.

Although a single generated step makes script replay more reliable, it is not able to replay asynchronous actions (such as pause and seek) that you may have performed while recording the script. The single generated step also does not replay the automatic requests that the client performs when Dynamic Stream is in use. If it is important to replay these asynchronous actions, you must record the Vuser script using the regular recording mode - not the simplified recording mode - and then manually modify the generated script as described below.

Modifying scripts to replay asynchronous user actions

If your Vuser script must be able to replay asynchronous actions that are performed while a streaming action is executed, you must record the Vuser script using the regular recording mode - not the simplified recording mode - and then manually modify the generated script. The modified script will include a combination of single streaming steps and the more verbose steps that are generated with regular recording.

Note: In this section, we will use the term *required user actions* to refer to the actions that must be performed while a video is streamed.

To create a script that can replay asynchronous behavior, first you record the script using the regular recording mode - not the simplified recording mode. Thereafter, identify the **flex_rtmp_send** steps that represent *required user actions*. Then replace the steps between the *required user actions* with single streaming functions. See the sections below for details.

Note: The modification procedure differs slightly between RTMP and RTMPT steps.

Modifying recorded Flex RTMP steps

When you use the regular recording mode, VuGen generates **flex_rtmp_receive** and **flex_rtmp_send** steps for all communication with the server. This ensures that user actions such as pause and seek, as well as automatic requests that the client performs when Dynamic Stream is in use, are included in the script. However, this method also captures less-necessary lines of code that are difficult to read and may not be reliable during replay of streaming actions.

Note: To activate the regular recording mode, clear the **Generate single step for RTMP/T stream handling** check box in the **Flex > RTMP** pane of the Recording Options dialog box.

Follow the instructions below to remove the unnecessary **flex_rtmp_receive** and **flex_rtmp_send** steps from your script.

1. Search for the **flex_rtmp_send** step that contains the initial play argument. For example:

   flex_rtmp_send("send_step2",

   "ConnectionID=10",

   "Snapshot=tRTMP6.inf",

MESSAGE,

...

...

MESSAGE,

...

...

"Argument=<arguments><string>play</string><number>0</number><null/>"

...

LAST);

2. Delete or comment out the **flex_rtmp_receive** steps that occur during streaming. For example:

```
//this is the start of the stream:
// flex_rtmp_receive("recv_step2",
// "ConnectionID=10",
// "Snapshot=tRTMP7.inf",
// CHANNEL,
// "ChunkStreamID=2",
// CHANNEL,
// "ChunkStreamID=2",
// CHANNEL,
// "ChunkStreamID=4",
// CHANNEL,
// "ChunkStreamID=2",
// LAST);
//
// flex_rtmp_receive("recv_step3",
// "ConnectionID=10",
// "Snapshot=tRTMP8.inf",
// CHANNEL,
// "ChunkStreamID=5",
// CHANNEL,
// ...
// ...
```

3. Remove the **flex_rtmp_send** steps that are not related to the required user actions, such as

"user control message" types. For example:

// flex_rtmp_send("send_step3",

// "ConnectionID=10",

// "Snapshot=tRTMP9.inf",

// MESSAGE,

// "DataType=user control message",

// "EventType=set buffer length",

// "MessageStreamID=1",

// "BufferLength=100",

// LAST);

4. When you find a **flex_rtmp_send** step that represents a required user action, do the following:
   a. Manually add a **flex_rtmp_receive_stream** step before the send step.

      - Make sure that the **ConnectionID** argument has the same value as the steps you removed above it.

      - The **Snapshot** argument is not relevant for the manually added step.

      - You can use the **ContinueToNexStepAfter = <msec>** argument to control the minimum play duration of the stream to download before continuing to the next step.

   b. Determine the **flex_rtmp_send** steps that represent the required user actions. These will likely include arguments such as **pauseRaw**, **pause**, **seek** and **play2** (for Dynamic Stream). For example:

      flex_rtmp_send("send_step5",

      "ConnectionID=10",

      "Snapshot=tRTMP62.inf",

      MESSAGE,

      "DataType=command message amf3",

      "ChunkStreamID=8",

      "MessageStreamID=1",

      "Argument=<arguments><string>pauseRaw</string><number>0</number><null/>"

      "<boolean>true</boolean><number>12000</number></arguments>",

      LAST);

   c. Determine whether there are some extra **flex_rtmp_send** steps that you can remove. For example, if you dragged a button to seek in the stream, subtle jerks in the motion may be recorded as separate pause and seek actions. In these cases, may not need all of them. Keep only those that describe the desired operations.

   d. Identify the **flex_rtmp_receive** step that indicates that the server has received the end of the user action. For example:

```
//this is the confirmation from the server on the "seek" command.
flex_rtmp_receive("recv_step55",
"ConnectionID=10",
"Snapshot=tRTMP68.inf",
CHANNEL,
"ChunkStreamID=2",
CHANNEL,
"ChunkStreamID=2",
LAST);
```

5. Repeat steps 2 - 4 for each set of unnecessary receive data and required user actions in your script.

For additional details on **flex_rtmp_receive_stream** including a complete example, see the Function Reference (**Help > Function Reference**).

Modifying recorded Flex RTMPT steps

When you use the regular recording mode, VuGen generates a **flex_rtmp_tunneled_send** step for all communication with the server. This ensures that user actions such as pause and seek, as well as automatic requests that the client performs when Dynamic Stream is in use, are included in the script. However, this method also captures less-necessary lines of code that are difficult to read and may not be reliable during replay of streaming actions.

Note: To activate the regular recording mode, clear the **Generate single step for RTMP/T stream handling** check box in the **Flex > RTMP** pane of the Recording Options dialog box.

Follow the instructions below to remove the unnecessary steps from your script.

1. Search for the **flex_rtmp_tunneled_send** step that contains the initial play argument. For example:

```
flex_rtmp_tunneled_send("send_step2",
"SessionID=1",
"Snapshot=t36.inf",
MESSAGE,
...
...
MESSAGE,
...
...
"Argument=<arguments><string>play</string><number>0</number><null/>"
...
LAST);
```

2. Remove **flex_rtmp_tunneled_send** steps that are not related to required user actions, such

as "user control message" types. For example:

```
// flex_rtmp_tunneled_send("send_step3",
// "SessionID=10",
// "Snapshot=t15.inf",
// MESSAGE,
// "DataType=user control message",
// "EventType=set buffer length",
// "MessageStreamID=1",
// "BufferLength=100",
// LAST);
```

3. When you find a **flex_rtmp_tunneled_send** step that represents a required user action, do the following:

   a. Add a **ContinueToNexStepAfter = <msec>** argument to the previous step. The **ContinueToNexStepAfter = <msec>** argument controls the minimum play duration of the stream to download before continuing to the next step. For example:

   ```
   flex_rtmp_tunneled_send("send_step2",
   "SessionID=1",
   "Snapshot=t36.inf",

   //Read the stream until at least 15 seconds of media have been downloaded

   "ContinueToNexStepAfter = 15000",
   MESSAGE,
   ...
   ...
   MESSAGE,
   ...
   ...
   "Argument=<arguments><string>play</string><number>0</number><null/>"
   ...
   LAST);
   ```

   b. Determine the **flex_rtmp_tunneled_send** steps that represent the required user actions. These will typically include arguments such as **pauseRaw**, **pause**, **seek** and **play2** (for Dynamic Stream). For example:

   ```
   flex_rtmp_tunneled_send("send_step5",
   "SessionID=10",
   "Snapshot=t16.inf",
   MESSAGE,
   "DataType=command message amf3",
   "ChunkStreamID=8",
   "MessageStreamID=1",
   "Argument=<arguments><string>pauseRaw</string><number>0</number><null/>"
   "<boolean>true</boolean><number>12000</number></arguments>",
   LAST);
   ```

   c. Determine whether there are extra **flex_rtmp_tunneled_send** steps that you can remove. For example, if you dragged a button to seek in the stream, subtle jerks in the motion may

be recorded as separate pause and seek actions. In these cases, you may not need all of them. Keep only those that describe the desired operations.

4. Repeat steps 2 - 3 for each set of unnecessary send data and required user actions in your script.

For additional details on the **flex_rtmp_tunneled_send** function, including a complete example, see the Function Reference (**Help > Function Reference**).

Live Streaming

VuGen's Flex protocol supports Adobe's Live Streaming. If VuGen detects a live stream while you record a Vuser script, VuGen adds '**ContinueToNextStepAfter**' and '**ContinueMode**' arguments to the generated **flex_rtmp_receive_stream** or **flex_rtmp_tunneled_send** function. These additional arguments enable the live stream to be accurately replayed. For details on these arguments, see the Function Reference (**Help > Function Reference**).

Note: The default value of the generated **ContinueToNextStepAfter** argument is the length of time (in milliseconds) for which the video was streamed while the Vuser script was recorded.

Externalizable Objects in Flex Scripts

When recording a Flex application, information is usually passed between the client and server using known serialization methods (AMF). If this is the case, VuGen creates a **flex_amf_call** and both the request and response are parsed.

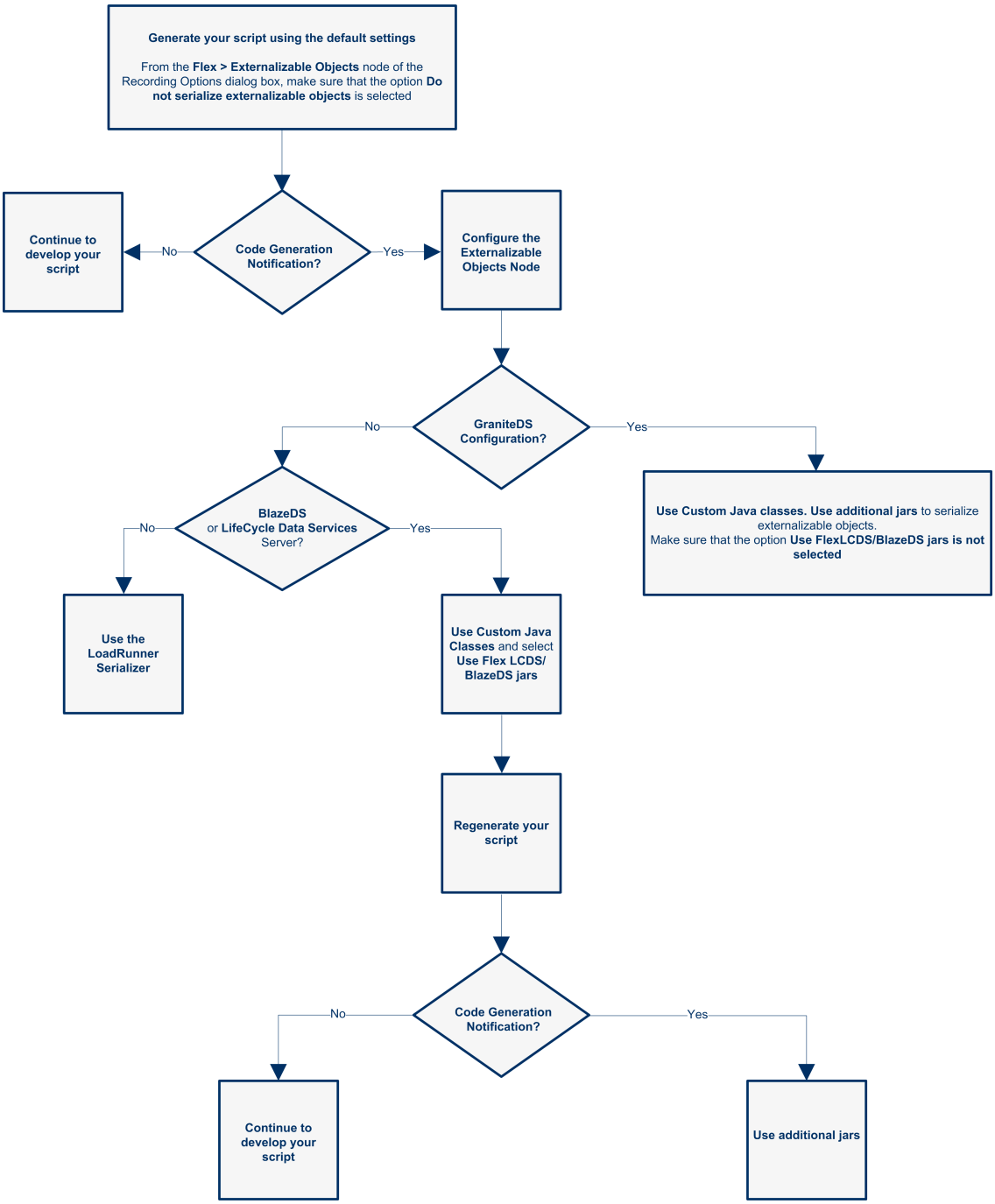
However, when a given AMF object uses a custom serialization method (externalizable), VuGen automatically issues a warning. This warning displays details about the exception, as well as recommended actions for resolving it.

The following are some examples of the exceptions that the generated script may include when an AMF object uses a custom serialization method:

- **Request and response not parsed.** This exception is automatically displayed in the Errors pane when the **Warnings** button is selected, and the **Define Available Categories** filter is set to **All** or **Code Generation Notification**. Details about the exception are listed, as well as recommended actions. For details, see "Errors Pane" on page 94.
- **Request parsed but response is not parsed.** VuGen generates a `IsParseResponse=No` statement. Additionally, VuGen issues a warning that is automatically displayed in the Errors pane when the **Warnings** button is selected, and the **Define Available Categories** filter is set to **All** or **Code Generation Notification**. The list of warnings displays details about the exception, as well as recommended actions. For details, see "Errors Pane" on page 94.

For details on configuring the **Recording Options > Flex > Externalizable Objects** Node, see "Flex > Externalizable Objects Node (Recording)" on page 288.

The following flowchart illustrates the steps to resolve externalizable objects in Flex scripts:



For details on how to serialize externalizable objects, see:

- "Flex > Externalizable Objects Node (Recording)" on page 288
- "How to Serialize Using External Java Serializer" on page 524
- "How to Serialize Scripts with the LoadRunner Serializer" on page 525

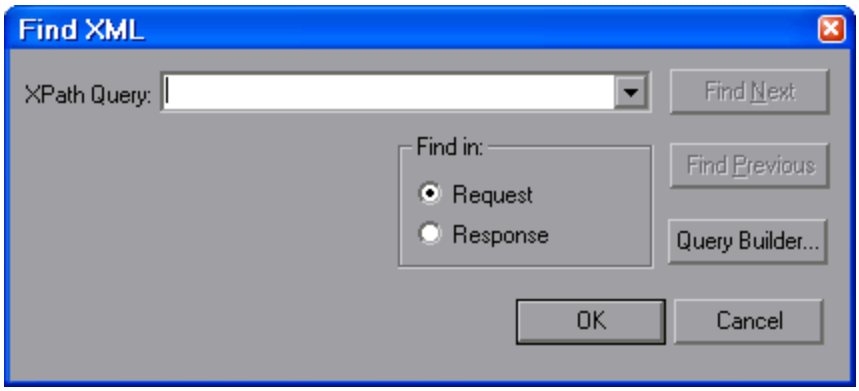
How to Query an XML Tree

VuGen provides a Query Builder that lets you create and execute queries on the XML.

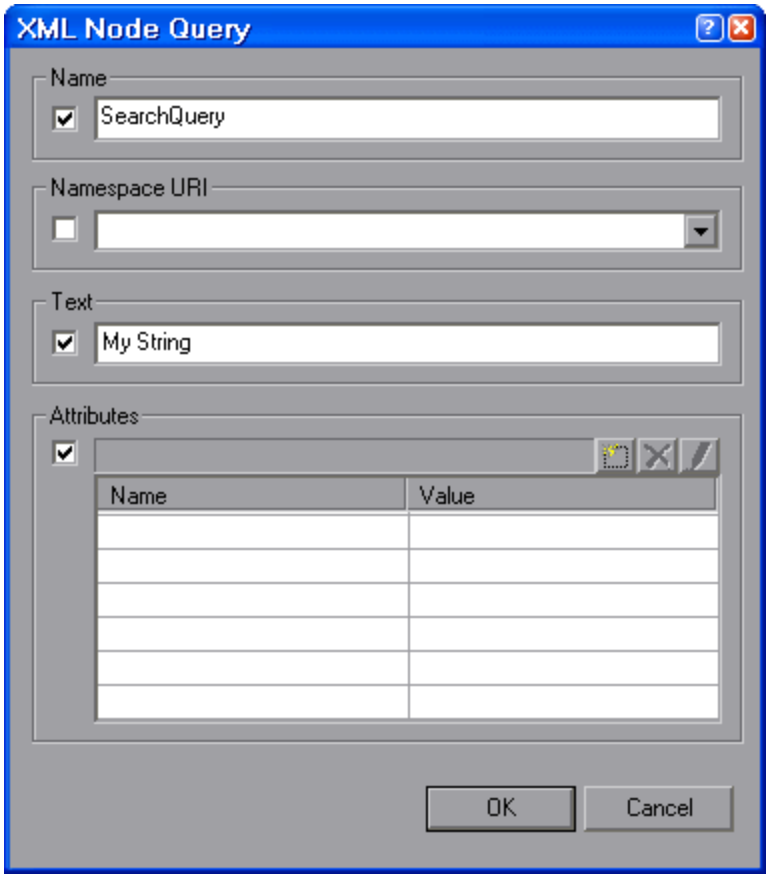
VuGen displays the XML code in an expandable tree. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

Perform a query

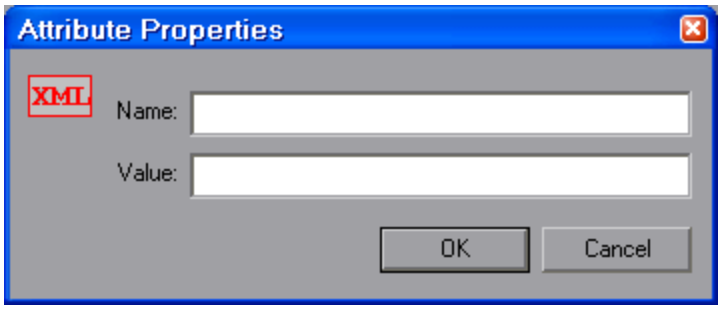
1. In the Snapshot pane, select the node that you want to search. Click the **Find XML** button. The Find XML dialog button opens.



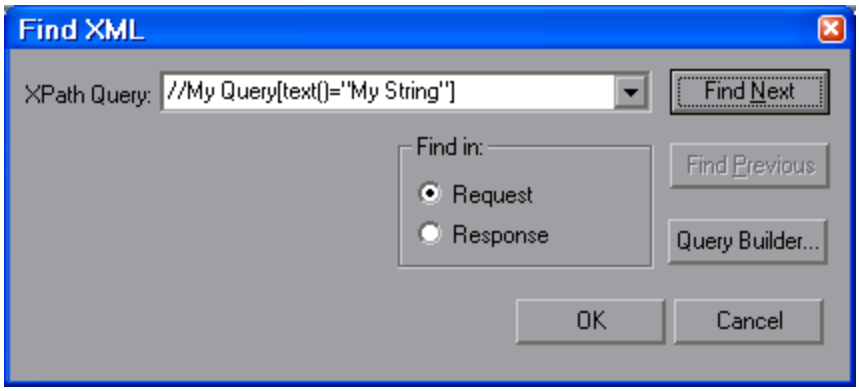
2. Select **Request** or **Response**. Enter an XPath query and click **OK**. To formulate a query, click **Query Builder** button. The XML Node Query dialog box opens.
3. Enable one or more items for searching.



4. Enable the **Name** section to search for the name of a node or element.
5. Enable the **Namespace URI** section to search for a namespace.
6. Enable the **Text** section to search for the value of the element indicated in the Name box.
7. Enable the **Attributes** section to search for an attribute.
8. Enter the search text in the appropriate boxes. To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.



9. Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the Find XML box.



10. Click **Find Next** to begin the search.

How to Serialize Using External Java Serializer

You can use the Java classes from the Flex server to serialize AMF messages in your script. This process has been simplified so that you need to include the application JAR files only if the AMF objects implement an externalizable interface.

Handle Externalizable Objects Using an External Serializer

1. In the **Recording Options > Flex > Externalizable Objects** node, select **Serialize objects using** and select **Custom Java Classes** from the drop-down menu.
2. Add the relevant files by using the **Add all classes in folder** or **Add JAR or Zip file** buttons. Add the following files:
   a. **For Adobe BlazeDS or Adobe LCDS**, add the following JAR files:
      - flex-messaging-common.jar

- flex-messaging-core.jar

b. Regenerate the script and note any errors. Open the recording options dialog box using the **Generation Options** button and add the necessary application JAR files.

3. Ensure that the added files exist in the same location both on the VuGen machine and on all load generators.

Notes and Limitations for the Java Serializer

- Supported JDK versions: 1.6 and earlier.
- Supported servers: Adobe BlazeDS and Adobe Livecycle DS.
- Microsoft .NET classes are not supported.
- During code generation VuGen performs a validity test of the request buffers by verifying that the buffer can be read and written using the provided jars. Failure in this validity test indicates that the classes are incompatible with LoadRunner.

How to Serialize Scripts with the LoadRunner Serializer

You can attempt to serialize externalizable objects using the LoadRunner serializer. Ensure that you have saved all open scripts because this option may result in unexpected errors or invalid steps.

Use the LoadRunner Serializer

1. Save all open scripts in VuGen.
2. In the **Recording Options > Flex > Externalizable Objects** node, select **Serialize objects using** and select **LoadRunner AMF serializer** from the Reference

Flex AMF Example Scripts

In the following AMF0 example, the **flex_amf_call** function accesses a gateway and sends a message to the server.

```
flex_amf_call("EchoAny",
       "Gateway=http://<host>/gateway.aspx",
       "Snapshot=t1.inf",
       "IsParseResponse=No",
       MESSAGE,
       "Method=EchoAMF.EchoAMF.EchoAny",
       "TargetObjectId=/1",
       BEGIN_ARGUMENTS,
             "<boolean>true</boolean>",
       END_ARGUMENTS,
       LAST);
```

In the following AMF3 example, the **flex_remoting_call** function sends the server an AMF call that can be serialized.

```
            flex_remoting_call(
            "product::getProductsByName",
             URL=http://<HOST>:<PORT>/amf;jsessionid=
{CorrelationParameter}",
             "Snapshot=t1.inf",
             "IsParseResponse=No",
             INVOCATION,
             "Target=/2",
             "Operation=getProductsByName",
             "Destination=product",
             "DSEndpoint=my-amf",
             "DSId=8E3759E5-E51A-3906-0EAB-6119CD1E26BF",
             "Argument="
                   "<arguments>"
                          "<string>A</string>"
                   "</arguments>",
             LAST);
```

For detailed syntax information on these functions, see the Function Reference (**Help > Function Reference**).

Setting the Flex Recording Mode

You can instruct VuGen how to generate a script from a Flash Remoting session using the Flex and Web Protocols.

Example

Use Web HTTP technology to generate **web_custom_request** functions with the Flash Remoting information.

```
web_url("flash",
    "URL=http://<HOST>:<PORT>/flash/",
    "Resource=0",
    "RecContentType=text/html",
    "Referer=",
    "Snapshot=t1.inf",
    "Mode=HTML",
    EXTRARES,
    "Url=movies/XMLExample.swf", "Referer=", ENDITEM,
    "Url=movies/JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);
web_link("Sample JavaBean Movie Source",
    "Text=Sample JavaBean Movie Source",
    "Snapshot=t2.inf",
    EXTRARES,
    "Url=XMLExample.swf", "Referer=", ENDITEM,
    "Url=JavaBeanExample.swf", "Referer=", ENDITEM,
    LAST);
web_custom_request("gateway",
```

```
    "URL=http://<HOST>:<PORT>/flashservices/gateway",
    "Method=POST",
    "Resource=0",
    "RecContentType=application/x-amf",
    "Referer=",
    "Snapshot=t3.inf",
    "Mode=HTML",
    "EncType=application/x-amf",
    "BodyBinary=\\x00\\x00\\x00\\x01\\x00\\x10amf_server_debug\\x01
                \\x00\\x00\\x00`\\x03\\x00\ncoldfusion\\x01\\x01\\x00
                \namfheaders\\x01\\x00\\x00\\x03amf\\x01\\x00\\x00
                \\x0Bhttpheaders\\x01\\x00\\x00\trecordset\\x01\\x01
                \\x00\\x05error\\x01\\x01\\x00\\x05trace\\x01\\x01
                \\x00\\x07m_debug\\x01\\x01\\x00\\x00\t\\x00\\x01
                \\x00/flashgateway.samples.FlashJavaBean.testDocument
                \\x00\\x02/1\\x00\\x00\\x004\n\\x00\\x00\\x00\\x01
                \\x0F\\x00\\x00\\x00*<TEST
message=\"test\"><INSIDETEST /></TEST>",
        LAST);
```

Flex Functions and Examples

When you record a Flex application, VuGen generates Flex Vuser script functions that emulate the application. The following functions represent some of the Flex remoting steps:

Function Name	Description
flex_amf_call	Sends an AMF request.
flex_amf_define_envelope_ header_set	Defines a set of envelope headers.
flex_amf_define_header_set	Defines a set of AMF headers.
flex_login	Logs on to a password-protected Flex application.
flex_logout	Logs off of a password-protected Flex application.
flex_ping	Checks if a Flex application is available.
flex_remoting_call	Invokes one or more methods of a server-side Remote object (RPC).

Example:

In the following example, **flex_ping** checks for the availability of a service. The **flex_remoting_call** function invokes the service remotely.

```
    flex_ping("1",
        "URL=http://<HOST>/weborb.aspx",
        "Snapshot=t6.inf",
        LAST);
    flex_remoting_call("getProductEdition::GenericDestination",
```

```
"URL=http://testlab1/weborb30/console/weborb.aspx",
"Snapshot=t1.inf",
INVOCATION,
"Target=/2",
"Operation=getProductEdition",
"Destination=GenericDestination",
"DSEndpoint=my-amf",
"Source=Weborb.Management.LicenseService",
"Argument=<arguments/>",
LAST);
```

RTMP Functions

Function Name	Description
flex_rtmp_connect	Connects a client to an RTMP server and sets connection options.
flex_rtmp_disconnect	Disconnects a client from an RTMP server
flex_rtmp_receive_stream	Receives streaming data from an RTMP server
flex_rtmp_receive	Receives responses from an RTMP server
flex_rtmp_send	Sends a request to an RTMP server

Example:

In the following example, **flex_rtmp_receive** receives data.

```
     flex_rtmp_receive("recv_step0",
"ConnectionID=19",
"Snapshot=tRTMP44.inf",
CHANNEL,
"ChunkStreamID=2",
CHANNEL,
"ChunkStreamID=2",
LAST);
```

RTMP Tunneled Functions

Function Name	Description
flex_rtmp_tunneled_connect	Connects a client to an RTMP server over HTTP.
flex_rtmp_tunneled_ disconnect	Disconnects a client from session over HTTP with an RTMP server
flex_rtmp_tunneled_send	Sends a request to an RTMP server over HTTP

Example:

In the following example, **flex_rtmp_tunneled_send** sends an RTMP tunneled request.

```
flex_rtmp_tunneled_send(
 "send_step0",
 "SessionID=0",
 "Snapshot=t30.inf",
 MESSAGE,
 "DataType=command message amf3",
 "ChunkStreamID=3",
 "MessageStreamID=0",
 "Argument="
       "<arguments>"
                              ...
       "</arguments>",
 LAST);
```

For detailed syntax information about all of the Flex functions, see the Function Reference (**Help > Function Reference**).

Java Protocol

Java Protocol Recording Overview

Using VuGen, you can record a Java application or applet. VuGen creates a pure Java script enhanced with Vuser API Java-specific functions. After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it. Once you verify that the script is functional, you incorporate it into a LoadRunner scenario or Business Process Monitor configuration.

When you create a script through recording and manual enhancements, all of the guidelines and limitations associated with Java Vuser scripts apply. In addition, any specific classes used in the script must be present on the machine executing the Vusers and indicated by the **classpath** environment variable. See "Java Protocol - Manually Programming Scripts" on page 544 for important information about function syntax and system configuration.

Before recording a CORBA session, verify that your application or applet functions properly on the recording machine.

Make sure that you have properly installed a JDK version from Sun on the machine running VuGen—JRE alone is insufficient. You must complete this installation before recording a script. Verify that the **classpath** and **path** environment variables are set according to the JDK installation instructions.

Note: When you load an applet or application from VuGen during recording, it may take several seconds longer than if you were to load it independent of VuGen.

VuGen provides a tool that enables you to convert a Vuser script created for Web, into Java. For more information, see "How to Convert Web Vuser Scripts into Java" on page 679.

After recording, you can enhance or modify the script with standard Java code using JDK libraries or custom classes.

After you prepare your script, you run it in standalone mode from VuGen. Sun's standard Java compiler, **javac.exe**, checks the script for errors and compiles it.

You integrate finished scripts into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the Function Reference, or *Performance Center* documentation.

Java Vuser Script Overview

When you record a session, VuGen logs all calls to the server and generates a script with functions. These functions describe all of your actions within the application or applet. The script also contains supplementary code required for proper playback, such as property settings, and naming service initialization (JNDI).

The recorded script is comprised of three sections:

- Imports
- Code
- Variables

The **Imports** section is at the beginning of the script. It contains a reference to all the packages required for compiling the script. The **Code** section contains the Actions class and the recorded code within the **init**, **actions**, and **end** methods. The **Variables** section, after the **end** method, contains all the type declarations for the variables used in the code.

After you finish recording, you can modify the functions in your script, or add additional Java or LoadRunner functions to enhance the script. Note that if you intend to run Java Vusers as threads, the Java code you add to your script must be thread-safe. For details about function syntax, see the Function Reference (**Help > Function Reference**). In addition, you can modify your script to enable it to run as part of another package. For more information, see "Compiling and Running a Script as Part of a Package" on page 547.

RMI over IIOP Overview

The **Internet Inter-ORB Protocol** (IIOP) technology was developed to allow implementation of CORBA solutions over the World Wide Web. IIOP lets browsers and servers exchange complex objects such as arrays, unlike HTTP, which only supports transmission of text.

RMI over IIOP technology makes it possible for a single client to access services which were only accessible from either RMI or CORBA clients in the past. This technology is a hybrid of the JRMP protocol used with RMI and IIOP used with CORBA. **RMI over IIOP** allows CORBA clients to access new technologies such as **Enterprise Java Beans** (EJB) among other J2EE standards.

VuGen provides full support for recording and replaying Vusers using the **RMI over IIOP** protocol. Depending on what you are recording, you can utilize VuGen's RMI recorder to create a script that will optimally emulate a real user:

- **Pure RMI client.** recording a client that uses native JRMP protocol for remote invocations
- **RMI over IIOP client.** recording a client application that was compiled using the IIOP protocol

instead of JRMP (for compatibility with CORBA servers).

Corba Recording Options

For recording a CORBA session, you need to set the following options in the Recording Options:

- JNDI
- Use DLL hooking to attach VuGen support

CORBA Application Vendor Classes

Running CORBA applications with JDK1.2 or later, might load the JDK internal CORBA classes instead of the specific vendor CORBA classes. To force the virtual machine to use the vendor classes, specify the following java.exe command-line parameters:

```
Visigenic 3.4
-Dorg.omg.CORBA.ORBClass=com.visigenic.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.visigenic.vbroker.orb.
     ORBSingleton
Visigenic 4.0
-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton
OrbixWeb 3.x
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.
     singletonORB
OrbixWeb 2000
-Dorg.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl
-Dorg.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.
     ORBSingleton
```

Recording RMI

Before recording an RMI session, verify that your application or applet functions properly on the recording machine.

Before you record, verify that your environment is configured properly. Make sure that the required classes are in the classpath and that you have a full installation of JDK. For more information on the required environment settings, see "How to Manually Create a Java Script" on page 547.

Recording a Jacada Vuser

The Jacada Interface Server provides an interface layer for mainframe applications. This layer separates the user interface from the application logic in order to insulate the organization from changes in standards and technologies. Instead of working with green-screen applications, the Jacada server converts the environment to a user friendly interface.

VuGen records Jacada's Java thin-client. To record communication with the Jacada server through the HTML thin-client, use the Web HTTP/HTML type Vuser. For more information, see "Web Protocols" on page 669.

Before replay, you must also download the **clbase.jar** file from the Jacada server. All classes used by the Java Vuser must be in the classpath—either set in the machine's classpath environment variable or in the **Classpath Entries** list in the **Classpath** node of the run-time settings.

During replay, the Jacada server may return screens from the legacy system, in a different order than they appear in the recorded script. This may cause an exception in the replay. For information on how to handle these exceptions, contact HP support.

Working with CORBA

CORBA-specific scripts usually have a well-defined pattern. The first section contains the ORB initialization and configuration. The next section indicates the location of the CORBA objects. The following section consists of the server invocations on the CORBA objects. The final section includes a shutdown procedure which closes the ORB. Note that pattern is not mandatory and that each one of these sections may appear multiple times within a script.

In the following segment, the script initializes an ORB instance and performs a bind operation to obtain a CORBA object. Note how VuGen imports all of the necessary classes.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;
public class Actions {
   // Public function: init
   public int init() throws Throwable {
   // Initialize Orb instance...
   MApplet mapplet = new MApplet("http://chaos/classes/", null);
   orb = org.omg.CORBA.ORB.init(mapplet, null);

   // Bind to server...
   grid = grid_dsi.gridHelper.bind("gridDSI", "chaos");
   return lr.PASS;
}
```

The `org.omg.CORBA.ORB` function makes the connection to ORB. Therefore, it should only be called once. When running multiple iterations, place this function in the **init** section.

In the following section, VuGen recorded the actions performed upon a grid CORBA object.

```
// Public function: action
public int action() throws Throwable {

   grid.width();
   grid.height();
   grid.set(2, 4, 10);
   grid.get(2, 4);
   return lr.PASS;
}
```

At the end of the session, VuGen recorded the shutdown of the ORB. The variables used throughout the entire recorded code appear after the **end** method and before the Actions class closing curly bracket.

```
// Public function: end
    public int end() throws Throwable {
        if (lr.get_vuser_id() == -1)
            orb.shutdown();
        return lr.PASS;
}
// Variable section
    org.omg.CORBA.ORB orb;
    grid_dsi.grid ;
}
```

Note that the ORB shutdown statement was customized for this product. This customization prevents a single Vuser's shutdown from shutting down all other Vusers.

Working with RMI

This section describes the elements of the Java Vuser script that are specific to RMI. RMI does not have constructs (as in CORBA)—instead it uses Serializable Java objects. The first section performs a Naming Registry initialization and configuration. The next section is generated when Java objects (both Remote and Serializable) are located and casted. The following section consists of the server invocations on the Java objects. In RMI there is no specific shutdown section (unlike CORBA). Note that objects might appear multiple times within the script.

The following segment locates a naming registry. This is followed by a lookup operation to obtain a specific Java object. Once you obtain the object, you can work with it and perform invocations such as **set_sum**, **increment**, and **get_sum**. The following segment also shows how VuGen imports all of the necessary RMI classes.

```
Import java.rmi.*;
Import java.rmi.registry.*;
:
:
// Public function: action
public int action() throws Throwable {

    _registry = LocateRegistry.getRegistry("localhost",1099);
    counter = (Counter)_registry.lookup("Counter1");
    counter.set_sum(0);
    counter.increment();
    counter.increment();
    counter.get_sum();
    return lr.PASS;
    }
:
```

When recording RMI Java, your script may contain several calls to **lr.deserialize**, which deserializes all of the relevant objects. The **lr.deserialize** calls are generated because the object being passed to the next invocation could not be correlated to a return value from any of the previous calls. VuGen therefore records its state and uses **lr.deserialize** call to represent these values during replay. The deserialization is done before VuGen passes the objects as parameters to

invocations. For more information, see "How to Correlate Scripts - Java Scripts - Serialization" on page 155.

Working with Jacada

The Actions method of a Java Vuser script using Jacada, has two main parts: properties and body. The properties section gets the server properties. VuGen then sets the system properties and connects to the Jacada server.

```
 // Set system properties...
        _properties = new Properties(System.getProperties());
        _properties.put("com.ms.applet.enable.logging", "true");
        System.setProperties(_properties);

        _jacadavirtualuser = new cst.client.manager.JacadaVirtualUser
();

        lr.think_time(4);
        _jacadavirtualuser.connectUsingPorts("localhost", 1100,
"LOADTEST", "", "", "");
     ...
```

The body of the script contains the user actions along with the exception handling blocks for the checkFieldValue and checkTableCell methods.

```
     l...
/*
try {
            _jacadavirtualuser.checkFieldValue(23, "S44452BA");
        }catch(java.lang.Exception e) {
             lr.log_message(e.getMessage());
        }
*/     l...
/*
try {
    _jacadavirtualuser.checkTableCell(41, 0, 0, "");
        }catch(java.lang.Exception e) {
             lr.log_message(e.getMessage());
        }
*/     l...
```

The **checkField** method has two arguments: field ID number and expected value. The **checkTableCell** method has four arguments: table ID, row, column, and expected value. If there is a mismatch between the expected value and the received value, an exception is generated.

By default, the try-catch wrapper blocks are commented out. To use them in your script, remove the comment markers.

In addition to the recorded script, you can add any of the Java Vuser API functions. For a list of these functions and information on how to add them to your script, see "Java Protocol - Manually Programming Scripts" on page 544.

Java Custom Filters - Overview

This section describes the background information necessary to create custom Java filters. For task details, see "How to Create a Custom Java Filter" on page 541.

When testing your Java application, your goal is to determine how the server reacts to client requests. When load testing, you want to see how the server responds to a load of many users. With VuGen's Java Vuser, you create a script that emulates a client communicating with your server.

VuGen provides filter files that define hooking properties for commonly used methods. There are filter definitions for RMI, CORBA, JMS, and JACADA protocols. You can also define custom filters.

When you record a method, the methods which are called from the recorded method either directly or indirectly, will not be recorded.

In order to record a method, VuGen must recognize the object upon which the method is invoked, along with the method's arguments. VuGen recognizes an object if it is returned by another recorded method provided that:

- The construction method of that object is hooked.
- It is a primitive or a built-in object.
- It supports a serializable interface.

You can create a custom filter to exclude unwanted methods. When recording a Java application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters for RMI, CORBA, JMS, and JACADA protocols were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your JAVA application's calls or exclude unnecessary calls. Custom JAVA protocols, proprietary enhancements and extensions to the default protocols, and data abstraction all require a custom filter definition.

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, VuGen allows you to record with a stack trace that logs all of the methods that were called by your application. In order to record with stack trace set the log level to **Detailed**.

Java Custom Filters - Determining which Elements to Include

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- **Top Down Approach.** An approach in which you include the relevant package and exclude

specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined layer which implements all client-server activity without involving any GUI elements.

- **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT packages and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- If you identify a non-client/server call in your script, exclude its method in the filter.
- During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen serializes it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by deserializing it.
- VuGen serializes objects passed as arguments that were not included by the filter. We recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the **lr.deserialize()** method in your script. For more information see "How to Correlate Scripts - Java Scripts - Serialization" on page 155.
- Exclude all activity which involves GUI elements.
- Add classes for utilities that may be required for the script to be compiled.

How to Record a Java Vuser Script

This task describes how to record a Java vuser script.

1. ### Prerequisites

   Make sure that you have properly installed a JDK version from Sun on the machine running the Vusers—JRE alone is insufficient. Verify that the classpath and path environment variables are set according to the JDK installation instructions. Before you replay a Vuser script, verify that your environment is configured properly for the JDK and relevant Java classes.

2. ### Create a new Java Record Replay Protocol script

   Select **File** > **New** and select **JAVA Record Replay** from the **Java** category.

3. ### Complete the Start Recording dialog box

   Enter the details of your application in the Start Recording dialog box. For user interface details, see "Record Dialog Box" on page 114.

4. ### Set the Recording Options

   In the Start Recording dialog box, click Options to open the Recording Options dialog box. In the **Recorder Options** node, the **Recorded Protocol** field configures the main protocol you will be recording. If you are recording more than one Java protocol, enter the additional protocols in the **Extension List** field.

How to Record Java Scripts Using Windows XP and 2000 Server

When recording on Windows XP and Windows 2000 servers, the Java plug-in may be incompatible with VuGen's recorder. To insure proper functionality, perform the following procedure after the installation of the java plug-in, before recording a script.

Configure Your Machine for Recording CORBA or RMI Sessions

1. Open the Java Plug-in from the Control Panel. Select **Start > Settings > Control Panel** and open the **Java Plug-in** component. The Basic tab opens.
2. Clear the **Enable Java Plug-In** check box and click **Apply**. Then, reselect the **Enable Java Plug-In** check box and click **Apply.**
3. Open the Browser tab. Clear the **Microsoft Internet Explorer** check box and click **Apply**. Then, reselect the **Microsoft Internet Explorer** check box and click **Apply.**

How to Run a Script as Part of a Package

This section is not relevant for Jacada type scripts.

When creating or recording a Java script, you may need to use methods from classes in which the method or class is protected. When attempting to compile such a script, you receive compilation errors indicating that the methods are not accessible.

To use the protected methods, add the Vuser to the package of required methods. At the beginning of your script, add the following line:

```
package a.b.c;
```

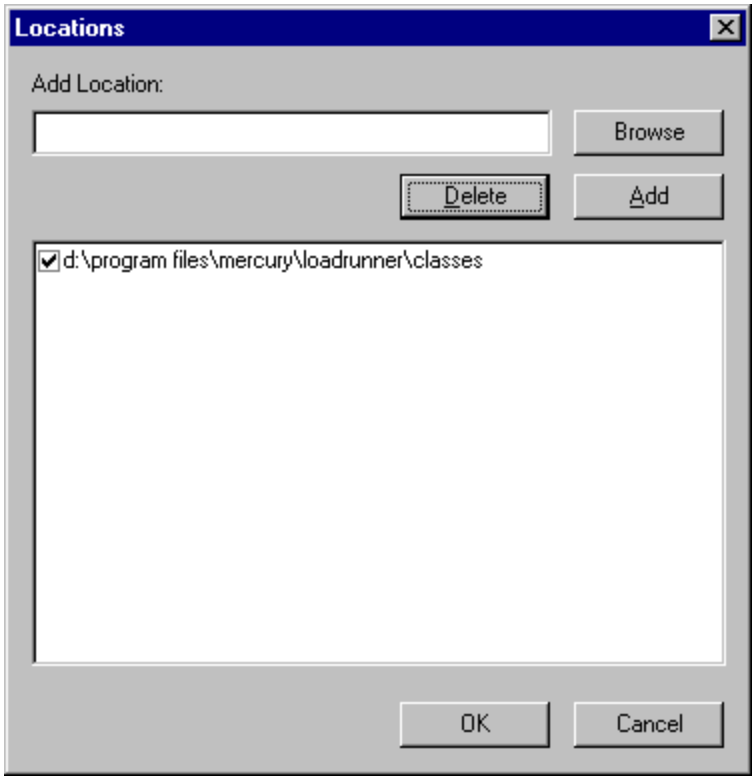
where **a.b.c** represents a folder hierarchy. VuGen creates the a/b/c folder hierarchy in the user folder and compiles the **Actions.java** file there, thus making it part of the package. Note that the **package** statement is not recorded—you need to insert it manually.

How to Manually Insert Java Methods

You use the Java Function navigator to view and add Java functions to your script. You can customize the function generation settings by modifying the configuration file. For more information, see "General > Script Node" on page 292.

Insert Java Functions

1. Click within your script at the desired point of insertion.
2. Select **Insert > Insert Java Function**. The Insert Java Function dialog box opens. The lower part of the dialog box displays a description of the Java object.
3. Click **Locations**. The Locations dialog box opens. By default, VuGen lists the paths defined in the CLASSPATH environment variable.



4. Click **Browse** to add another path or archive to the list. To add a path, select **Browse > Folder**. To add an archive (**jar** or **zip**), select **Browse > File**. When you select a folder or a file, VuGen inserts it in the **Add Location** box.
5. Click **Add** to add the item to the list.
6. Repeat steps 4 and 5 for each path or archive you want to add.
7. Select or clear the check boxes to the left of each item in the list. If an item is checked, its members will be listed in the Java Class navigator.
8. Click **OK** to close the Locations dialog box and view the available packages.
9. Click the plus and minus signs to the left of each item in the navigator, to expand or collapse the trees.
10. Select an object and click **Paste**. VuGen places the object at the location of the cursor in the script. To paste all the methods of a class into your script, select the class and click **Paste**.
11. Repeat the previous step for all of the desired methods or classes.
12. Modify the parameters of the methods. If the script generation setting **DefaultValues** is set to **true**, you can use the default values inserted by VuGen. If **DefaultValues** is set to **false**, you must add parameters for all methods you insert into the script.

    In addition, modify any return values. For example, if your script generated the following statement `"(String)=LavaVersion.getVersionId();"`, replace (String) with a string type variable.
13. Add any necessary statements to your script such as imports or Vuser API Java functions (described in "Java Protocol - Manually Programming Scripts" on page 544).

14. Save the script and run it from VuGen.

How to Manually Configure Script Generation Settings

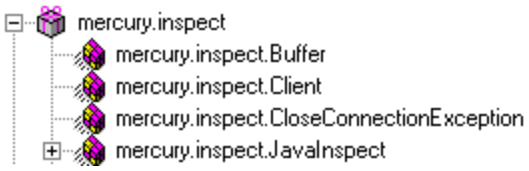
You can customize the way the navigator adds methods to your script.

To view the configuration setting, open the **jquery.ini** file in VuGen's dat folder.

```
[Display]
FullClassName=False
[Insert]
AutoTransaction=False
DefaultValues=True
CleanClassPaste=False
```

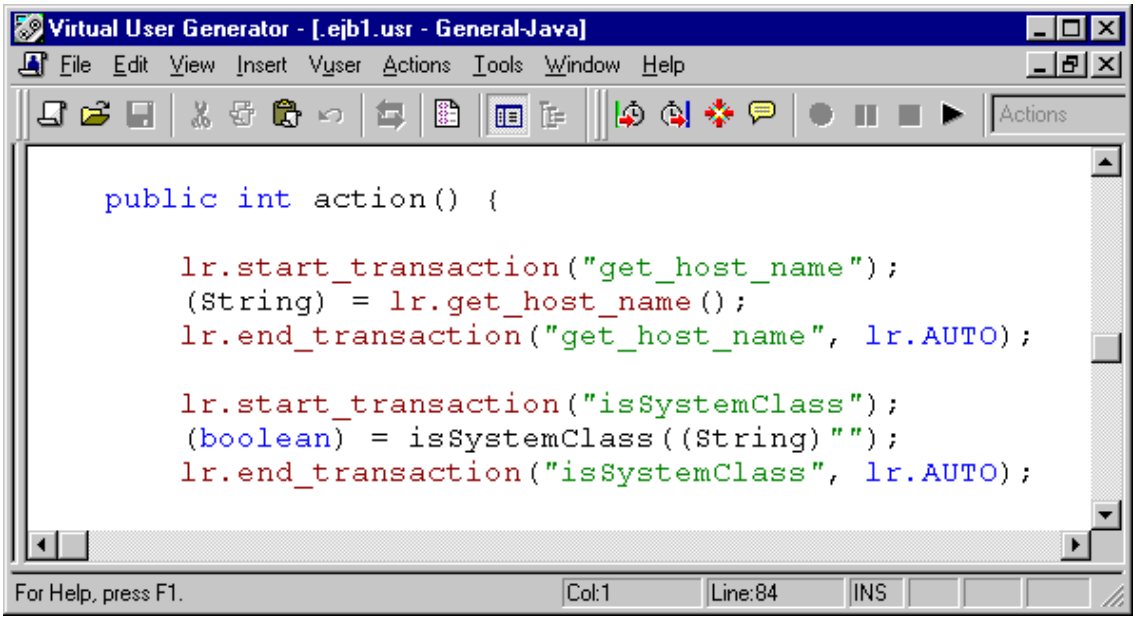
Class Name Path

The **FullClassName** option displays the complete package and class name in the Java Function navigator. This option does not affect the way the functions are added into the script—it only affects the way the classes are displayed in the navigator. By default, this option is set to false. If your packages have many classes and you are unable to view the package and class names at the same time, you should enable this option.

FullClassName enabled	FullClassName disabled
	

Automatic Transactions

The **AutoTransaction** setting creates a Vuser transaction for all methods. When you enable this option, VuGen automatically encloses all Java methods with **lr.start_transaction** and **lr.end_transaction** functions. This allows you to individually track the performance of each method. This option is disabled by default.



Default Parameter Values

The **DefaultValues** setting includes default values for all methods you paste into your script. This option is enabled by default and inserts a null for all objects. If you disable this option, you must manually insert parameter values for all functions in the script. The following table illustrates the DefaultValues flag enabled and disabled.

DefaultValues enabled	DefaultValues disabled

`lr.message((String)"");`
`lr.think_time((int)0);`
`lr.enable_redirection((boolean) false);`
`lr.save_data((byte[])null, (String) "");`
	`lr.message((String));`
	`lr.think_time((int));`
	`lr.enable_redirection ((boolean));`
	`lr.save_data((byte[]), (String));`

Class Pasting

The **CleanClassPaste** setting pastes a class so that it will compile cleanly: with an instance returning from the constructor, with default values as parameters, and without a need for import statements. Using this option, you will most likely be able to run your script without any further modifications. If you disable this option (default), you may need to manually define parameters and include import statements. Note that this setting is only effective when you paste an entire class into your script—not when you paste a single method.

The following segment shows the toString method pasted into the script with the CleanClassPaste option enabled.

```
_class.toString();
     // Returns: java.lang.String
```

The same method with the CleanClassPaste option disabled is pasted as follows:

```
(String) = toString();
```

The next segment shows the **NumInserter** Constructor method pasted into the script with the CleanClassPaste option enabled.

```
utils.NumInserter _numinserter = new utils.NumInserter
                ((java.lang.String)"", (java.lang.String)"",
(java.lang.String)""...);
// Returns: void
```

The same method with the CleanClassPaste option disabled is pasted as:

```
new utils.NumInserter((String)"", (String)"", (String)"",...);
```

How to Create a Custom Java Filter

This task describes how to create a custom Java filter. For background information, see "Java Custom Filters - Overview" on page 535.

For details of the Hook File structure, see "Hook File Structure" on next page.

When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Note: If you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this, is that if you re-record a script after modifying the filters, it will overwrite all manual changes.

Define a Custom Java Filter

1. Create a new filter based on one of the built-in filters by modifying the **user.hooks** file which is located in the product's **classes** folder. For structural details about the user.hook file, see "Hook File Structure" on next page.
2. Open the Recording Options (Ctrl+F7) and select the **Log Options** node. Select the Log Level to **Detailed**.
3. Record your application. Click **Start Record** (Ctrl + R) to begin and **Stop** (Ctrl + F5) to end.
4. View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain and correlate, you should customize the script's filter.
5. Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) or by viewing a Stack Trace of the script.
6. Set the filter to include the relevant methods. For more information, see "Java Custom Filters - Determining which Elements to Include" on page 535.
7. Record the application again. You should always rerecord the application after modifying the filter.
8. Repeat steps 4 through 7 until you get a simple script which can be maintained and correlated.

9. Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information about the built-in correlation mechanism, see "How to Correlate Scripts - Java Scripts - Serialization" on page 155 and "How to Correlate Scripts - Java Scripts - Serialization" on page 155.

Note: Do not modify any of the other .hooks file as it might damage the VuGen recorder.

Adding custom hooks to the default recorder is a complicated task and should be considered thoroughly as it has both functional and performance consequences.

Incorrect hooking definitions can lead to incorrect scripts, slow recording, and application freeze-up.

Hook File Structure

The following section describes the structure of a typical .hooks file:

```
[Hook-Name]
class      = MyPackage.MyClass
method     = MyMethod
signature  = ()V
ignore_cl =
ignore_mtd =
ignore_tree =
cb_class   = mercury.ProtocolSupport
cb_mtd =
general_cb = true
deep_mode = soft | hard
make_methods_public = true | false
lock = true | false
```

The hook files are structured as .ini files where each section represents a hook definition. Regular expressions are supported in some of the entries. Any entry that uses regular expression must start with a '!'.

Hook-Name

Specifies the name of this section in the hooks file. Hook-Name must be unique across all hooks files. A good practice is to give the fully qualified class name and method. For example:

[javax.jms.Queue.getQueueName]

Class

A fully qualified class name. Regular expression can be used to include several classes from the same package, a whole package, several packages, or any class that matches a name. For example:

Class = !javax\.jms\.*

Method

The simple name of the method to include. Regular expressions can be used to include more than one method from the class. For example:

Method = getQueueName

Signature

The standard Java internal type signature of the method. To determine the signature of a method, run the command j`avap -s class-name` where `class name` is the fully qualified name of the class. Regular expressions can be used to include several methods with the same name, but with different arguments. For example:

Signature = !.*

ignore_cl

A specific class to ignore from the classes that match this hook. This can be a list of comma separated class names. Each item in the list can contain a regular expression. If an item in the list contains a regular expression, prepend a '!' to the class name. For example:

Ignore_cl = !com.hp.jms.Queue,!com\.hp\..*

ignore_mtd

A specific method to ignore. When the loaded class method matches this hook definition, this method will not be hooked. The method name must be the simple method name followed by the signature (as explained above). To ignore multiple methods, list them in a comma separated list. To use a regular expression, prepend a '!' to the method name. For example:

Ignore_cl = open, close

ignore_tree

A specific tree to ignore. When the name of the class matches the ignore tree expression, any class that inherits from it will not be hooked, if it matches this hooks definition. To ignore multiple trees, list them in a comma separated list. To use a regular expression, prepend a '!' to the class name. This option is relevant only for hooks that are defined as deep.

cb_class

The callback class that gets the call from the hooked method. It should always be set to **mercury.ProtocolSupport**.

cb_mtd

A method in the callback class that gets the call from the hooked method. If omitted, it uses the default, **general_rec_func**. For cases where you just need to lock the subtree of calls, use **general_func** instead.

general_cb

The general callback method. This value should always be set to **true**.

Deep_mode

Deep mode refers to classes and interfaces that inherit or implement the class or interface that the hook is listed for. The inherited classes will be hooked according to the type of hook: **Hard**, **Soft**, or **Off**.

- **Hard.** Hooks the current class and any class that inherits from it. If regular expressions exist, they are matched against every class that inherits from the class in the hook definition. Interface inheritance is treated the same as class inheritance.

- **Soft.** Hooks the current class and any class that inherits from it, only if the methods are overridden in the inheriting class. If the hook lists an interface, then if a class implements this interface those methods will be hooked. If they exist in classes that directly inherit from that class they will also be hooked. However, if the hook lists an interface and a class implements a second interface that inherits from this interface, the class will not be hooked.

Note: Regular expressions are not inherited but converted to actual methods.

- **Off.** Only the class listed in the hook definition and the direct inheriting class will be hooked. If the hook lists an interface, only classes that directly implement it will be hooked.

make_methods_public:

Any method that matches the hook definition will be converted to public. This is useful for custom hooks or for locking a sub tree of calls from a non-public method.

Note that this applies only during record. During replay, the method will use the original access flags. In the case of non-public methods, it will throw `java.lang.VerifyError`.

Lock

When set to **true**, it locks the sub tree and prevents the calling of any method originating from the original method.

When set to **false**, it will unlock the sub tree, record any method originating from the current method (if it is hooked), and invoke the callback.

Java Icon Reference List

The following table describes the icons that represent the various Java objects:

Icon	Item	Example
	Package	java.util
	Class	public class Hashtable extends java.util.Dictionary implements java.lang.Cloneable, java.io.Serializable
	Interface Class (gray icon)	public interface Enumeration
	Method	public synchronized java.util.Enumeration keys ()
	Static Method (yellow icon)	public static synchronized java.util.TimeZone getTimeZone
	Constructor Method	public void Hashtable ()

Java Protocol - Manually Programming Scripts

Manually Programming Java Scripts - Overview

To prepare Vuser scripts using Java code, use the **Java** type Vusers. This Vuser type supports Java on a protocol level. The Vuser script is compiled by a Java compiler and supports all of the standard Java conventions. For example, you can insert a comment by preceding the text with two forward slashes "//".

"Java Protocol" on page 529 explains how to create a script through recording using the **Java Record Replay** Vuser. To prepare a Java coded script through programming, see the following sections.

The first step in creating a Java compatible Vuser script, is to create a new Vuser script template of the type **Java Vuser**. Then, you program or paste the desired Java code into the script template. You can add Java Vuser functions to enhance the script and parameterize the arguments to use different values during iterations.

The Java Vuser script runs as a scalable multi-threaded application. If you include a custom class in your script, make sure that the code is thread-safe. Code that is not thread-safe may cause inaccurate results. For code that is not thread-safe, run the Java Vusers as processes. This creates a separate Java Virtual Machine for each process, resulting in a script that is less scalable.

After you prepare a script, run it as a standalone test from VuGen. A Java compiler (Sun's javac), checks it for errors and compiles the script.

After you create a script, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the Function Reference, *Performance Center* documentation.

Java Protocol Programming Tips

When programming a Java Vuser script, you can paste ready-made code segments into scripts or import ready-made classes in order to invoke their methods. If Vusers need to run as threads under the Controller (for scalability reasons), you need to make sure that all of the imported code is thread-safe.

Thread-safety is often difficult to detect. A Java Vuser may run flawlessly under VuGen and under the Controller with a limited number of Vusers. However, problems may then occur with a large number of Vusers. Code that is not thread-safe is usually the result of static class member usage as shown in the following example:

```
import lrapi.*;
public class Actions
{
   private static int iteration_counter = 0;
    public int init() {
        return 0;
    }
    public int action() {
        iteration_counter++;
        return 0;
    }
    public int end() {
```

```
        lr.message("Number of Vuser iterations: "+iteration_counter);
        return 0;
    }
}
```

When you run one Vuser, the **iteration_counter** member determines the number of iterations that were executed. When multiple Vusers run together as threads on a single virtual machine, the static class member **iteration_counter** is shared by all threads, resulting in an incorrect counting. The total number of all Vusers iterations is counted.

If code is known to be non thread-safe and you still want to import it into your script, you can run the Vusers as processes. For more information on running Vusers as threads or processes, see "Run-Time Settings" on page 332

When you run a basic Java Vuser script, it usually consists of a single thread—the main thread. Only the main thread can access the Java Vuser API. If a Java Vuser spawns secondary worker threads, using the Java API may cause unpredictable results. Therefore, we recommend that you use the Java Vuser API only in the main thread. Note that this limitation also affects the **lr.enable_redirection** function.

The following example illustrates where the LR API may and may not be used. The first log message in the execution log indicates that the value of flag is false. The virtual machine then spawns a new thread set_thread. This thread runs and sets flag to true, but will not issue a message to the log, even though the call to lr.message exists. The final log message indicates that the code inside the thread was executed and that flag was set to true.

```
boolean flag = false;
public int action() {
   lr.message("Flag value: "+flag);
   Thread set_thread = new Thread(new Runnable();{
        public void run() {
           lr.message("LR-API NOT working!");
           try {Thread.sleep(1000);} catch(Exception e) {}
           flag = true;
        }
   });
   set_thread.start();
   try {Thread.sleep(3000);} catch(Exception e) {}
   lr.message("Flag value: "+flag);
   return 0;
}
```

Running Java Vuser Scripts

Java Vuser scripts differ from C Vuser scripts in that they are first compiled and then executed; C Vuser scripts are interpreted. VuGen locates the **javac** compiler from within the JDK installation and compiles the Java code inside the script. This stage is indicated by the **Compiling...** status message in the bottom of the VuGen window. If errors occur during compilation, they are listed in the execution log. To go to the code in your script that caused the error, double-click on the error message containing the line number of the error. Fix the error and run the script again.

If the compilation succeeds, the status message **Compiling...** changes to **Running...** and VuGen begins to execute the script. When you run the script again, VuGen runs the script without recompiling it, provided that no changes were made to the script. To debug your script further, you can use breakpoints and animated run type execution using the step option.

Note: If you are making calls to JNDI extensions within your script, you may encounter problems trying to run your Vusers as **threads**. This happens because JNDI requires each thread to have its own context class loader. In order to run as threads, instruct each Vuser to run with its own context class loader, by adding the following line to the beginning of the **init** section:

```
DummyClassLoader.setContextClassLoader();
```

Compiling and Running a Script as Part of a Package

When creating a Java Vuser script, you may need to use methods in other classes in which the class or method is protected. If you try to compile this type of script, you will receive errors in the compilation stage indicating that the methods are inaccessible. To make sure that your script can access these methods, insert the package name containing these methods at the top of the script, just as you would do in a standard Java program— <package_name>. In the following example, the script defines the just.do.it package which consists of a path:

```
package my.test;
import lrapi.*;
public class Actions
{
     :
}
```

In the above example, VuGen automatically creates the **my/test** folder hierarchy under the Vuser folder, and copies the **Actions.java** file to **my/test/Actions.java**, allowing it to compile with the relevant package. Note that the package statement must be the first line in the script, similar to Java (excluding comments).

How to Manually Create a Java Script

This task describes how to manually create and edit a custom Java script.

1. **Create a new script**
   a. Open VuGen.
   b. Select **File > New** or click the **New** button. The New Virtual User dialog box opens.
   c. Select **Custom > Java Vuser** from the Select Vuser type list, and click **OK**. VuGen displays a blank Java Vuser script.
   d. Click the **Actions** section in the left frame to display the **Actions** class.
2. **Insert your code into the script**
   After generating an empty template, you can insert the desired Java code. When working with

this type of Vuser script, you place all your code in the Actions class. To view the Actions class, click **Actions** in the left pane. VuGen displays its contents in the right pane.

```
import lrapi.*;
public class Actions
{
    public int init() {
        return 0;
    }
    public int action() {
        return 0;
    }
    public int end() {
        return 0;
    }
}
```

The Actions class contains three methods: init, action, and end. The following table shows what to include in each method and when each method is executed.

Script method	Used to emulate...	Is executed when...
init	a login to a server	the Vuser is initialized (loaded)
action	client activity	the Vuser is in "Running" status
end	a log off procedure	the Vuser finishes or is stopped

Init Method

Place all the login procedures and one-time configuration settings in the init method. The init method is only executed once—when the Vuser begins running the script. The following sample init method initializes an applet. Make sure to import the **org.omg.CORBA.ORB** function into this section, so that it will not be repeated for each iteration.

```
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import lrapi.lr;
// Public function: init
    public int init() throws Throwable {
    // Initialize Orb instance...
    MApplet mapplet = new MApplet("http://chaos/classes/", null);
    orb = org.omg.CORBA.ORB.init(mapplet, null);
...
```

Action Method

Place all Vuser actions in the action method. The action method is executed according to the number of iterations you set in the runtime settings. For more information on the iteration settings, see "Run-Time Settings" on page 332 The following sample action method retrieves and prints the Vuser ID.

```
 public int action() {
    lr.message("vuser: " + lr.get_vuser_id() + " xxx");
```

```
        return 0;
    }
```

End Method

In the **end** method, place the code you want the Vuser to execute at the end of the script, such as logging off from a server, cleaning up the environment, and so forth.

The end method is only executed once—when the Vuser finishes running the script. In the following example, the end method closes and prints the end message to the execution log.

```
public int end() {
        lr.message("End");
        return 0;
    }
```

3. Insert additional LoadRunner API functions

VuGen provides a specific Java API for Java Vuser scripts. These functions are all static methods of the lrapi.lr class.

The Java API functions are classified into several categories: Transaction, Command Line Parsing, Informational, String, Message, and Run-Time functions.

For further information about each of these functions, see the Function Reference (**Help > Function Reference**). Note that when you create a new Java Vuser script, the import lrapi.* is already inserted into the script.

4. Insert additional Java functions

To use additional Java classes, import them at the beginning of the script as shown below.

Remember to add the classes folder or relevant jar file to the classpath. Make sure that the additional classes are thread-safe and scalable.

```
import java.io.*;
import lrapi.*;
public class Actions
{
...
}
```

5. Add script enhancements

You add script enhancements such as rendezvous points, transactions, and output messages. For more information, see "How to Enhance a Java Script" on next page.

6. Set the Java environment

Before running your Java Vuser script, make sure that the environment variables, path and classpath, are properly set on all machines running Vusers:

- To compile and replay the scripts, you must have complete JDK installation, either version 1.1 or 1.2, or 1.3. The installation of the JRE alone is not sufficient. It is preferable not to have more than one JDK or JRE installation on a machine. If possible, uninstall all unnecessary versions.
- The **PATH** environment variable must contain an entry for **JDK/bin**.

- For JDK 1.1.x, the **CLASSPATH** environment variable must include the **classes.zip** path, (**JDK/lib** subfolder) and all of the VuGen classes (**classes** subfolder).
- All classes used by the Java Vuser must be in the classpath—either set in the machine's **CLASSPATH** environment variable or in the **Classpath Entries** list in the Classpath node of the run-time settings.

How to Enhance a Java Script

This task describes how to enhance custom Java scripts.

Inserting Transactions

You define transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified requests. These requests can be short or complex tasks. When working with LoadRunner, you can analyze the performance per transaction during and after the scenario run, using online monitor and graphs.

You can also specify a transaction status: lr.PASS or lr.FAIL. You can let the Vuser automatically determine if the transaction was successful, or you can incorporate it into a conditional loop. For example, in your code you can check for a specific return code. If the code is correct, you issue a lr.PASS status. If the code is wrong, you issue an lr.FAIL status.

Mark a transaction

1. Insert **lr.start_transaction** into the script, at the point where you want to begin measuring the timing of a task.
2. Insert **lr.end_transaction** into the script, at the point where you want to stop measuring the task. Use the transaction name as it appears in the **lr.start_transaction** function.
3. Specify the desired status for the transaction: lr.PASS or lr.FAIL.

```
public int action() {
for(int i=0;i<10;i++)
   {
    lr.message("action()"+i);
    lr.start_transaction("trans1");
    lr.think_time(2);
    lr.end_transaction("trans1",lr.PASS);
   }
    return 0;
}
```

Inserting Rendezvous Points

The following section does not apply to the *HP Business Availability Center*.

To emulate heavy user load on your client/server system, you synchronize Vusers to perform a task at exactly the same moment by creating a rendezvous point. When a Vuser arrives at the rendezvous point, it is held by the Controller until all Vusers participating in the rendezvous arrive.

You designate the meeting place by inserting a rendezvous function into your Vuser script.

Insert a Rendezvous Point

- Insert an lr.rendezvous function into the script, at the point where you want the Vusers to perform a rendezvous.

```
public int action() {
  for(int i=0;i<10;i++)
   {
    lr.rendezvous("rendz1");
    lr.message("action()"+i);
    lr.think_time(2);
   }
   return 0;
}
```

Obtaining Vuser Information

You can add the following functions to your Vuser scripts to retrieve Vuser information:

lr.get_attrib_string	Returns a string containing command line argument values or runtime information such as the Vuser ID or the load generator name.
lr.get_group_name	Returns the name of the Vuser's group.
lr.get_host_name	Returns the name of the load generator executing the Vuser script.
lr.get_master_host_name	Returns the name of the machine running the LoadRunner Controller or Business Process Monitor.
lr.get_scenario_id	Returns the ID of the current scenario. (LoadRunner only)
lr.get_vuser_id	Returns the ID of the current Vuser. (LoadRunner only)

In the following example, the lr.get_host_name function retrieves the name of the computer on which the Vuser is running.

```
String my_host = lr.get_host_name();
```

For more information about the above functions, see the Function Reference (**Help > Function Reference**).

Issuing Output Messages

When you run a scenario, the Controller Output window displays information about script execution. You can include statements in a Vuser script to send error and notification messages to the Controller. The Controller displays these messages in the Output window. For example, you could insert a message that displays the current state of the client application. You can also save these messages to a file.

Note: Do not send messages from within a transaction. Doing so lengthens the transaction execution time and may skew the actual transaction results.

You can use the following message functions in your Vuser script:

lr.debug_message	Sends a debug message to the Output window.
lr.log_message	Sends a message to the Vuser log file.
lr.message	Sends a message to a the Output window.
lr.output_ message	Sends a message to the log file and Output window with location information.

In the following example, **lr.message** sends a message to the output indicating the loop number:

```
for(int i=0;i<10;i++)
    {
     lr.message("action()"+i);
     lr.think_time(2);
    }
```

For more information about the message functions, see the Function Reference (**Help > Function Reference**).

You can instruct the Vusers to redirect the Java standard output and standard error streams to VuGen's Execution log. This is especially helpful when you need to paste existing Java code or use ready-made classes containing **System.out** and **System.err** calls in your Vuser scripts. In the execution log, standard output messages are colored blue, while standard errors are shown in red.

The following example shows how to redirect specific messages to the standard output and standard error using lr.enable_redirection:

```
lr.enable_redirection(true);
System.out.println("This is an informatory message...");  //
Redirected
System.err.println("This is an error message...");  // Redirected
lr.enable_redirection(false);
System.out.println("This is an informatory message...");  // Not
redirected
System.err.println("This is an error message...");  // Not redirected
```

Note: When you set **lr.enable_redirection** to **true**, it overrides all previous redirections. To restore the former redirections, set this function to **false**.

For additional information about this function, see the Function Reference (**Help > Function Reference**).

Emulating User Think Time

The time that a user waits between performing successive actions is known as the think time. Vusers use the lr.think_time function to emulate user think time. In the following example, the Vuser waits two seconds between loops:

```
for(int i=0;i<10;i++)
    {
     lr.message("action()"+i);
```

```
    lr.think_time(2);
}
```

You can use the think time settings as they appear in the script, or a factor of these values. To configure how Vusers handle think time functions, open the runtime settings dialog box. For more information, see "Run-Time Settings" on page 332.

For more information about the lr.think_time function, see the Function Reference (**Help > Function Reference**).

Handling Command Line Arguments

You can pass values to a Vuser script at runtime by specifying command line arguments when you run the script. You insert command line options after the script path and filename in the Controller or Business Process Monitor. There are three functions that allow you to read the command line arguments, and then to pass the values to a Vuser script:

lr.get_attrib_double	Retrieves double precision floating point type arguments
lr.get_attrib_long	Retrieves long integer type arguments
lr.get_attrib_string	Retrieves character strings

Your command line should have the following format, where the arguments and their values are listed in pairs after the script name:

```
script_name   - argument argument_value   -argument argument_value
```

The following example shows the command line string used to repeat script1 five times on the machine pc4:

```
script1  -host pc4  -loop 5
```

For more information on the command line parsing functions, see the Function Reference (**Help > Function Reference**). For more information on how to insert the command line options, see the *LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

Java over HTTP Protocol

Java over HTTP Protocol Overview

The Java over HTTP protocol is designed to record java-based applications and applets. It produces a Java language script using web functions. This protocol is distinguished from other Java protocols in that it can record and replay Java remote calls over HTTP.

Note that Java over HTTP supports asymmetric Java object traffic. This means that object serialization traffic is recognized even when it is on only one side of the communication. This occurs when the request is serialization and the response is plain HTTP, or vice versa.

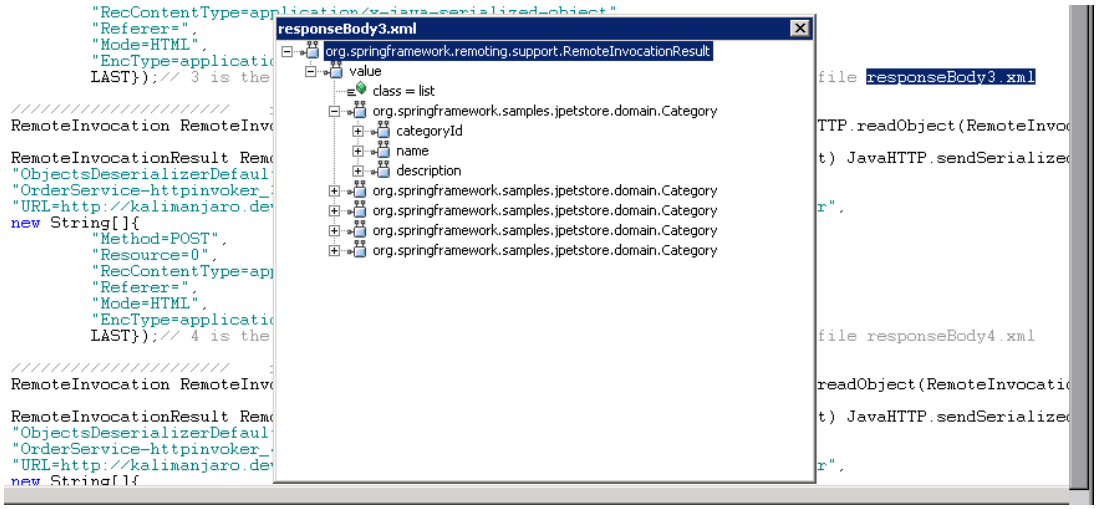
Viewing Responses and Requests in XML Format

Note that this topic applies to the Java over HTTP protocol only.

For each request and response, you can view the corresponding XML that represents the binary java object during the recording phase.

View XML data

1. Locate the target request or response section in the code. Right-click the commented **RequestBodyX.xml** or **ResponseBodyX.xml**.
2. Select **View XML**. The XML is displayed in a separate window.



How to Record with Java over HTTP

To record with Java over HTTP, you must specify which .jar files to use in order to deserialize the recorded data.

This topic describes how to locate the relevant .jar files and add them to the classpath.

Recording Java Applets

If your application uses Java Applets, you need to find the relevant .jar files and enable them in the classpath.

1. Clear the JAR cache by selecting **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Delete Files**.
2. Open your application and perform a few business processes to repopulate the JAR cache with .jar files from your application. When you are finished, close your application.
3. Select **Control Panel > Java > General Tab > Temporary Internet Files > View**. This lists the JAR cache and should contain only the .jar files used by your application.
4. Download the files. Try the options below in the order in which they appear. When you succeed, proceed to the next step to add the .jar files to the classpath.

a. Option 1: For each .jar file, go to the listed URL and download the file. If you cannot download one or more of the .jar files, continue with the next option.

b. Option 2: Clear the cache again by selecting **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Delete Files**. Open your application again and perform a few business processes. Do not close your application. Open the Java Console. There should be a message for each .jar file telling you the location it is stored in a temporary file on your computer. The files are usually hashed and don't have .jar extensions. Change the name (including changing each extension to .jar) and copy the file to a known location.

c. Option 3: If the files don't show up in the Java console, locate the temporary folder as listed in **Control Panel > Java > General Tab > Temporary Internet Files > Settings > Location**. Open the specified location and rename all the files in the sub-folders to .jar. Do not rename all the files in the main folder.

5. Add the .jar files to the classpath in the **Recording Options > Java Environment Settings > Classpath** node. For more information, see "Java > Classpath Node" on page 301.

Recording Local Java Applications

If you are recording a local Java application (not an applet), all of the .jar files already exist on your computer.

1. Look in the batch file that launched the application. All of the .jar files that are referenced should be added to the classpath.
2. If you cannot locate or understand the batch file, add all of the .jar files from the application folder and sub-folders to the classpath.
3. Add the .jar files to the classpath in the **Recording Options > Java Environment Settings > Classpath** node. For more information, see "Java > Classpath Node" on page 301.

How to Debug Java over HTTP scripts

This task describes how to debug Java over HTTP Vuser scripts by comparing the request and response data from the record and replay stages.

1. ### Add arguments to the VM Param Node

   Select **Replay > Run-Time Settings > Java VM** node. In the **Additional VM Parameters** field, enter the following string:

   ```
   -DdumpServerRequests=true -DdumpServerResponses=true
   ```

2. ### Compare record and replay data

   In the Solution Explorer, right-click the script name and select **Open Script Folder**. The data from the recording phase is in the main folder. The data from the replay phase is in the replay folder.

   The files that follow the format RequestBodyX contain the request data. The files that follow the format ResponseBodyX contain the response data.

   To compare the record and replay data for the purposes of debugging, compare the files with identical names from the recording and replay phases. For example, compare the RequestBody1 file from the main folder (recording phase) to the RequestBody1 file from the

replay folder. Normally, the files should be identical. Cases where the files are not identical may indicate problems in the script.

3. **Remove arguments before load testing**

   Return to the Java VM node and the items you added to the Additional VM Parameters field.

How to Insert Parameters into Java over HTTP Scripts

Parameter functions can be added for each response or request body text in a specific location. This location is indicated by a blank line, usually one to two lines below the start of the response or request body. In the example below, parameter functions can be added to the blank lines in each requestBody section.

```
//////////////////////    requestBody2.xml    //////////////////////
RemoteInvocation RemoteInvocation_getUsernameList2 =
    (RemoteInvocation) JavaHTTP.readObject(RemoteInvocationBA0);
//INSERT PARAMETERIZATION AND CORRELATION CODE HERE
RemoteInvocationResult RemoteInvocationResult_ArrayList2 =
    (RemoteInvocationResult) JavaHTTP.sendSerialized(RemoteInvocation_getUsernameList2, 2,
"ObjectsDeserializerDefaultImpl",
"OrderService-httpinvoker",
"URL=http://kalimanjaro.devlab.ad:8080/jpetstore/remoting/OrderService-httpinvoker",
new String[]{
        "Method=POST",
        "Resource=0",
        "RecContentType=application/x-java-serialized-object",
        "Referer=",
        "Mode=HTML",
        "EncType=application/x-java-serialized-object",
        LAST});// 2 is the number of the header file, record time response is at file responseBody2.xml

//////////////////////    requestBody3.xml    //////////////////////
RemoteInvocation RemoteInvocation_getCategoryList3 =
    (RemoteInvocation) JavaHTTP.readObject(RemoteInvocationBA1);
//INSERT PARAMETERIZATION AND CORRELATION CODE HERE
RemoteInvocationResult RemoteInvocationResult_ArrayList3 =
    (RemoteInvocationResult) JavaHTTP.sendSerialized(RemoteInvocation_getCategoryList3, 3,
"ObjectsDeserializerDefaultImpl",
"OrderService-httpinvoker_2",
"URL=http://kalimanjaro.devlab.ad:8080/jpetstore/remoting/OrderService-httpinvoker",
new String[]{
        "Method=POST",
        "Resource=0",
        "RecContentType=application/x-java-serialized-object",
        "Referer=",
        "Mode=HTML",
        "EncType=application/x-java-serialized-object",
        LAST});// 3 is the number of the header file, record time response is at file responseBody3.xml
```

Java over HTTP - Troubleshooting and Limitations

This section describes troubleshooting and limitations for the Java over HTTP protocol.

Limitations

- JDK 1.5 or higher is required.
- Lazy evaluating objects are not supported, for example hibernate in lazy mode.
- If there are stateful serialization mechanisms on the application server, this can interfere with LoadRunner deserialization and result in unserialized data and unexpected errors.
- The following menu items are not available for this protocol:
  - Insert > New Step / Start Transaction / End Transaction / Rendezvous

Disable Exception Error Checking

If you are receiving exception errors and you are sure that the error is irrelevant, VuGen allows you to disable all such error messages. To do this, select **Replay > Run-Time Settings > Java VM** node. In the **Additional VM Parameters** field, and append the following string to the end of the current entry:

```
-DvalidateServerResponse=false
```

Additionally, you can change the error checking behavior of a specific step by adding a closing argument to the sendSerialized function in script view. For more information, see the Function Reference.

Cannot Correlate Private Object Members

When you need to correlate or parameterize data that is a private member of an object, you can use the **lrapi.lr2.fieldSetter** and **lrapi.lr2.fieldGetter** functions.

```
 RemoteInvocation RemoteInvocation2 = (RemoteInvocation)
JavaHTTP.readObject(RemoteInvocationBA0);
            RemoteInvocation.methodName="applyToSchool";
            Student student=RemoteInvocation.arguments[0];

            Map grades=lr2.fieldGetter(student,"grades");//grades is a
private member of Student
            grades.put("Math","95");
            lr2.fieldSetter(student,"super.name","Tom");
             //Student class inherits the name field from Person. name
field is a string
            lr2.fieldSetter(student,"super.ID","98764321");
             //Student class inherits the ID field from Person. ID
field is an int
            RemoteInvocationResult RemoteInvocationResult_ArrayList2 =
             (RemoteInvocationResult) JavaHTTP.sendSerialized
(RemoteInvocation2, 2,
             "ObjectsDeserializerDefaultImpl",....
```

LDAP Protocol

LDAP Protocol Overview

LDAP, the Lightweight Directory Access Protocol, is a protocol used to access a folder listing. The LDAP folder is composed of many LDAP entries. Each LDAP entry is a collection of attributes with a name, called a distinguished name (DN). For more information about DN, see "Defining Distinguished Name Entries" on page 559.

LDAP folder entries are arranged in a hierarchical structure that reflects political, geographic, and/or organizational boundaries. Entries representing countries appear at the top of the tree. Below them are entries representing states or national organizations. Below them might be entries representing people, organizational units, printers, documents, or just about anything else.

VuGen records communication over LDAP servers. It creates a Vuser script, with functions that emulate your actions. This includes logging in and out of the server, adding and deleting entries, and querying an entry.

LDAP Protocol Example Script

All LDAP functions come in pairs—one for global sessions and one where you can indicate a specific session. To apply the action to all sessions, use the version without the **ex** suffix. To apply the action to a specific session, use the version with the session identifier with the **ex** suffix. For example, **mldap_logon** logs on to the LDAP server globally, while **mldap_logon_ex** logs on to the LDAP server for a specific session.

In the following example, the user logs on to an LDAP server, ldap1. It adds an entry and then renames the OU attribute from Sales to Marketing.

```
Action()
{
// Logon to the LDAP server

mldap_logon("Login",
                "URL=ldap://johnsmith:tiger@ldap1:80",

                LAST);

// Add an entry for Sally R. Jones

mldap_add("LDAP Add",

          "DN=cn=Sally R. Jones,OU=Sales, DC=com",

          "Name=givenName", "Value=Sally", ENDITEM,

          "Name=initials", "Value=R", ENDITEM,

          "Name=sn", "Value=Jones", ENDITEM,

          "Name=objectClass", "Value=contact", ENDITEM,

          LAST);

// Rename Sally's OU to Marketing

mldap_rename("LDAP Rename",

             "DN=CN=Sally R. Jones,OU=Sales,DC=com",

             "NewDN=OU=Marketing",

             LAST);

// Logout from the LDAP server

mldap_logoff();
    return 0;
}
```

Defining Distinguished Name Entries

The LDAP API references objects by its **distinguished name** (DN). A DN is a sequence of **relative distinguished names** (RDN) separated by commas.

An RDN is an attribute with an associated value in the form attribute=value. The attribute names are not case-sensitive. The following table lists the most common RDN attribute types.

String	Attribute Type
DC	domainComponent
CN	commonName
OU	organizationalUnitName
O	organizationName
STREET	streetAddress
L	localityName
ST	stateOrProvinceName
C	countryName
UID	userid

The following are examples of distinguished names:

```
DN=CN=John Smith,OU=Accounting,DC=Fabrikam,DC=COM
DN=CN=Tracy White,CN=admin,DC=corp,DC=Fabrikam,DC=COM
```

The following table lists reserved characters that cannot be used in an attribute value.

Character	Description
	space or # character at the beginning of a string
	space character at the end of a string
,	comma
+	plus sign
"	double quote
\	backslash
<	left angle bracket
>	right angle bracket
;	semicolon

To use a reserved character as part of an attribute value, you must precede it with an escape character, a backslash (\). If an attribute value contains other reserved characters, such as the equal sign (=) or non-UTF-8 characters, you must encode it in hexadecimal format—a backslash followed by two hex digits.

The following are examples of DNs that include escaped characters. The first example is an organizational unit name with an embedded comma; the second example is a value containing a carriage return.

```
DN=CN=Bitwise,OU=Docs\, Support,DC=Fabrikam,DC=COM
DN=CN=Before\0DAfter,OU=Test,DC=North America,DC=Fabrikam,DC=COM
```

LDAP Connection Options

Using the **mldap_logon[_ex]** function, you control the way you login to the LDAP server.

When specifying the URL of the LDAP server, you specify how to connect and with what credentials.

When specifying the server's URL, use the following format:

```
ldap[s][username:[password]@][server[:port]]
```

The following table shows several examples of connections to LDAP servers.

Syntax	Description
ldap://a:b@server.com:389	Connects to the server (to 389 port) and then binds with username "a" , password "b"
ldap://:@server.com	Connects to server (to default unsecured port 389) then binds anonymously with a NULL username and password
ldaps://a:@server.com	Connects to server (to default secured port 636) and then binds with username "a", password ""
ldap://@server.com, ldap://server.com	Connects to server without binding
ldap://a:b@	Binds with username "a", password "b, executing a bind on the existing session without reconnecting
ldap://:@	Binds anonymously with a NULL username and password (executes bind on existing session without reconnecting)

You can also specify LDAP modes or SSL certificates using the following optional arguments:

- **Mode.** The LDAP call mode: *Sync* or *Async*
- **Timeout.** The maximum time in seconds to search for the LDAP server
- **Version.** The version of the LDAP protocol version 1,2, or 3
- **SSLCertDir.** The path to the SSL certificates database file (cert8.db)
- **SSLKeysDir.** The path to the SSL keys database file (key3.db)

- **SSLKeyNickname.** The SSL key nickname in the keys database file
- **SSLKeyCertNickname.** The SSL key's certificate nickname in the certificates database file
- **SSLSecModule.** The path to the SSL security module file (secmod.db)
- **StartTLS.** Requires that the StartTLS extension's specific command must be issued in order to switch the connection to TLS (SSL) mode

For detailed information about these arguments, see the Function Reference (**Help > Function Reference**).

Mailing Service Protocols

Mailing Service Protocols Overview

The Mailing Service protocols emulate users working with email clients, viewing and sending emails. The following mailing services are supported:

- Internet Messaging (IMAP)
- MS Exchange (MAPI)
- Post Office Protocol (POP3)
- Simple Mail Transfer Protocol (SMTP)

The mail protocols support both record and replay, with the exception of MAPI that supports only replay.

IMAP Protocol Overview

IMAP Vuser script functions record the Internet Mail Application Protocol.

Each IMAP function begins with an **imap** prefix. For detailed syntax information on these functions, see the Function Reference (**Help > Function Reference**).

In the following example, the **imap_create** function creates several new mailboxes: Products, Solutions, and FAQs.

```
Actions()
{
    imap_logon("ImapLogon",
            "URL=imap://johnd:letmein@exchange.mycompany.com",
            LAST);
    imap_create("CreateMailboxes",
            "Mailbox=Products",
            "Mailbox=Solutions",
            "Mailbox=FAQs",
            LAST);
    imap_logout();
    return 1;
}
```

MAPI Protocol Overview

MAPI Vuser script functions generate activity to and from an MS Exchange server. Each MAPI

function begins with a **mapi** prefix. For detailed syntax information on these functions, see the Function Reference (**Help > Function Reference**). Note that recording of Vuser scripts is not supported for the MAPI protocol.

Note: To run MAPI scripts, you must define a mail profile on the machine running the script. For example, install Outlook Express, set it as the default mail client, and create a mail account. Alternatively, install Microsoft Outlook, set it as the default mail client, create a mail account and create a mail profile. To create a mail profile in Microsoft Outlook, select **Settings > Control Panel > Mail** > **Show Profiles** and add a mail profile.

In the following example, the **mapi_send_mail** function sends a sticky note through an MS Exchange server.

```
Actions()
{
    mapi_logon("Logon",
            "ProfileName=John Smith",
            "ProfilePass=Tiger",
            LAST);
    //Send a Sticky Note message
    mapi_send_mail("SendMail",
            "To=user1@techno.merc-int.com",
            "Cc=user0002t@techno.merc-int.com",
            "Subject=<GROUP>:<VUID> @ <DATE>",
            "Type=Ipm.StickyNote",
            "Body=Please update your profile today.",
            LAST);
    mapi_logout();
    return 1;
}
```

POP3 Protocol Overview

POP3 Vuser script functions emulate actions using the Post Office Protocol, POP3. Each function begins with a **pop3** prefix. For detailed syntax information on these functions, see the Function Reference (**Help > Function Reference**).

In the following example, the **pop3_retrieve** function retrieves five messages from the POP3 server.

```
Actions()
{
pop3_logon("Login", "
        URL=pop3://user0004t:my_pwd@techno.merc-int.com",
        LAST);
// List all messages on the server and receive that value
totalMessages = pop3_list("POP3", LAST);
// Display the received value (It is also displayed by the pop3_list
function)
lr_log_message("There are %d messages.\r\n\r\n", totalMessages);
// Retrieve 5 messages on the server without deleting them
```

```
pop3_retrieve("POP3", "RetrieveList=1:5", "DeleteMail=false", LAST);
pop3_logoff();
    return 1;
}
```

SMTP Protocol Overview

SMTP Vuser script functions emulate the Single Mail Transfer Protocol traffic. Each SMTP function begins with an **smtp** prefix. For detailed syntax information on these functions, see the Function Reference (**Help > Function Reference**).

In the following example, the **smtp_send_mail** function sends a mail message, through the SMTP mail server, techno.

```
Actions()
{
    smtp_logon("Logon",
               "URL=smtp://user0001t@techno.merc-int.com",
               "CommonName=Smtp Test User 0001",
               NULL);
    smtp_send_mail(    "SendMail",
                    "To=user0002t@merc-int.com",
                    "Subject=MIC Smtp: Sample Test",
                    "MAILOPTIONS",
                    "X-Priority: 3",
                    "X-MSMail-Priority: Medium",
                    "X-Mailer: Microsoft Outlook Express
5.50.400\r\n",
                    "X-MimeOLE: By Microsoft MimeOLE V5.50.00\r\n",
                    "MAILDATA",
                    "MessageText="
                        "Content-Type: text/plain;\r\n"
                        "\tcharset=\"iso-8859-1\"\r\n"
                        "Test,\r\n"
                        "MessageBlob=16384",
                 NULL);
    smtp_logout();
    return 1;
}
```

Mobile Protocols Overview

The VuGen Mobile protocols expand LoadRunner's capability of recreating user activity from mobile applications, both native[1] and browser-based[2]. With this solution you can:

[1] A mobile application, such as a new service, where the application resides on the device, but communicates with the server at various intervals.
[2] A browser-based application that has been configured for the display of the mobile device.

- Simulate users working on mobile devices
- Create scripts based on the recording of mobile devices or emulators

Protocol Options for Mobile

You can record user activity on a mobile device and generate scripts in VuGen using one of two protocols:

- **Mobile App (HTTP/HTML):**]: A protocol enabling you to develop scripts using mobile devices or device emulators communicating with servers over HTTP. You can record network traffic into a capture file (PCAP file) and create a VuGen script. Additionally, you can use a mobile emulator on your VuGen machine to develop your scripts.
- **Mobile TruClient:** A protocol enabling you to record user activity in browser-based mobile applications using Ajax TruClient technology. The Ajax TruClient browser is modified to emulate the display of your mobile browser.

The following table can help you determine which protocol is most suitable for your needs:

Application Type	Mobile TruClient	Mobile App (HTTP/HTML)
Browser-based applications	✓	✓
Native applications		✓

Network Speed Simulation

Network Simulation for mobile protocols models the behavior of the cellular network. This enables you to test an applications taking into consideration end-to-end response time from device to server. Three configuration options are available:

Maximum Bandwidth

This option is provided in cases where you do not wish to emulate a cellular network. This is the default setting.

Standard Bandwidth

Network Speed Simulation for mobile protocols provides predefined settings suitable for known cellular networks:

- General Packet Radio Service (GPRS)
- Enhanced Data rates for GSM Evolution (EDGE)
- Universal Mobile Telecommunications System (UMTS)
- High-Speed Downlink Packet Access (HSDPA)
- High-Speed Downlink Packet Access Phase 2 (HSDPA phase 2)
- High-Speed Uplink Packet Access (HSUPA)

Each network type has both a maximum and expected rate. The maximum rate represents the technology's best case performance rate while the expected rate more accurately reflects real time performance.

Custom Bandwidth

Network Speed Simulation enables you to set custom download and upload speeds, defined in bits. You can set either a single value or range for either upload or download speed. A case where this option would be useful is when you have expected network speed from your cellular provider for a specific area.

For details, see "Network > Speed Simulation Node" below.

Network > Speed Simulation Node

This dialog box enables you to configure bandwidth for the mobile protocol.

To access	**VuGen > Run-time settings > Network > Speed Simulation**
Important information	This option is only available for mobile protocols. "Network Speed Simulation" on previous page

User interface elements are described below:

UI Element	Description
Use maximum bandwidth	Vusers run at the maximum bandwidth that is available over the network. **Default value**: enabled.
Use standard bandwidth	Select a bandwidth associated with a cellular technology.
Use custom bandwidth	**Download speed:**Indicate a custom download speed in bits. Can be defined as a range or single value. **Upload speed:**Indicate a custom upload speed in bits. Can be defined as a range or single value. **Note:** • If you select custom bandwidth and both upload speed and download speed are left blank, the default setting of maximum bandwidth will be used. • If you select custom bandwidth and either upload speed or download speed are left blank, the empty value is automatically set to the defined value.

Recording Methods

The Mobile App (HTTP/HTML) protocol provides two methods for script generation:

- **Create a Script With a Capture File**

  You can analyze a capture file that was created with either an external capture file utilities, such as Wireshark, or VuGen's Mobile Sniffer Agent. The **Recording Wizard** > **Analyze Traffic** option enables you to point to the capture file, specify the source or destination IP address to include, and generate the script.

  For details see:

  "Recording Traffic into a Capture (Sniffer) File" below

  "Analyzing Traffic" on page 571

  "Recording Wizard" on page 579

- **Create a Script Using an Emulator**

  For many mobile devices, there are third party emulators that you can install on your computer. Once installed, you can use the **Emulate Record** method to record and generate a script.

  For details see:

  "Recording with Emulation" on page 571

  "Recording Wizard" on page 579

Recording Traffic into a Capture (Sniffer) File

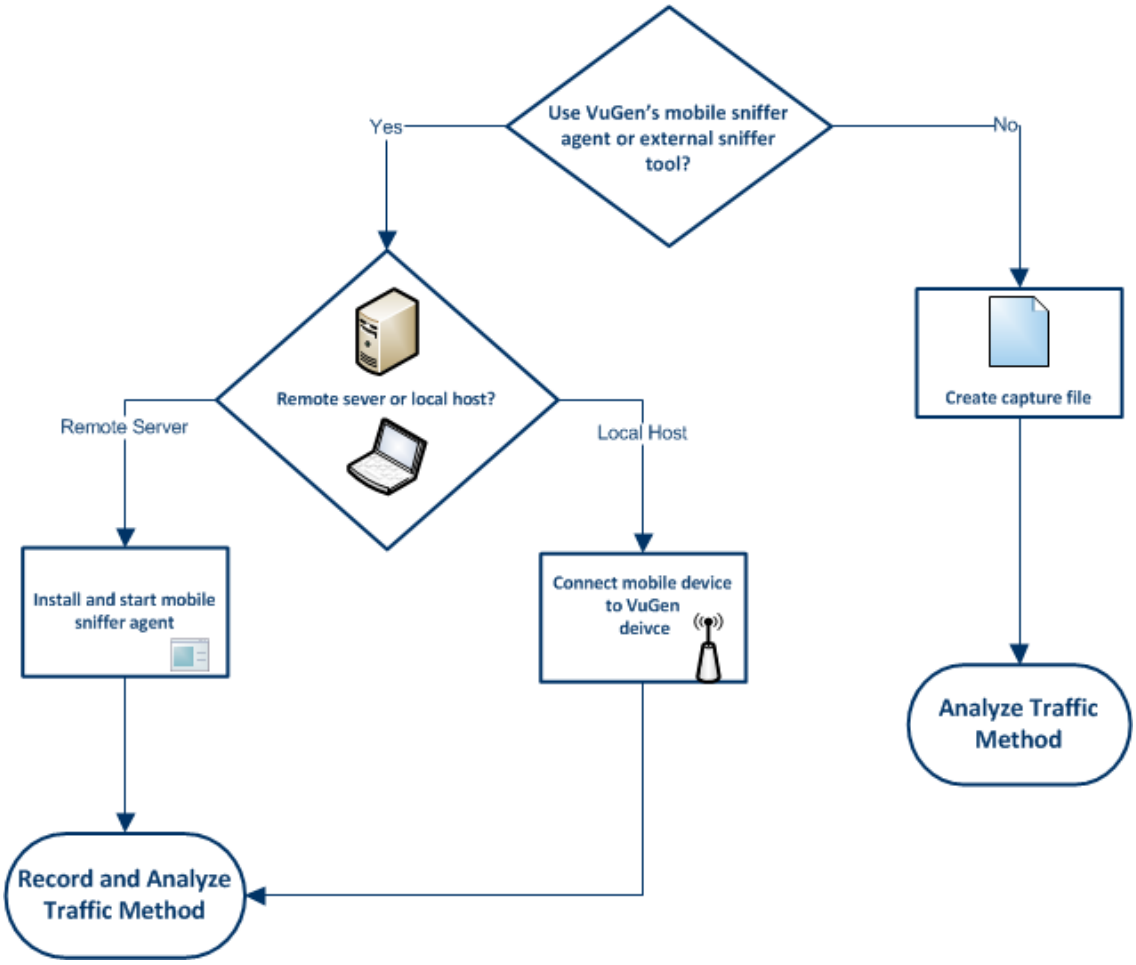
Recording application traffic to a capture file is effective when you are unable to record an application using VuGen as is the case with mobile applications. A capture file is a trace file containing a log of all TCP traffic over the network. Using a sniffer application, you obtain a dump of all of the network traffic. The sniffer captures all of the events on the network and saves them to a capture file. To generate a smaller, more manageable script, try to capture the network traffic only for the time that you perform actions in your application.

Depending on your OS and device, you have many options on where and how to record a capture file. The table below lists the some of the locations and their related advantages and disadvantages:

Capture File Source	Supported by Mobile Sniffer Agent	Advantages	Disadvantages
Mobile device		Record the real traffic that comes from the device	Requires administrator permissions on most devices

Capture File Source	Supported by Mobile Sniffer Agent	Advantages	Disadvantages
VuGen with configured "hot spot[1]"	✓	• Easy and intuitive • Administrator permission not an issue	• Not all devices support “ad hoc” • Captures WiFi only (Not capturing cellular network traffic)
Server	✓	Record real traffic (both WiFi and cellular network traffic)	• Requires administrator permissions • Requires installation of a software on the server.

You have the flexibility to record traffic with LoadRunner's mobile sniffer agent or record traffic with an external sniffer agent. The following flow chart illustrates the workflow for both methods:



[1] A WiFi connection

Record Traffic with VuGen's Mobile Sniffer Agent

Recording traffic on remote servers

To record traffic on a remote server you first must manually install the mobile sniffer agent on your server by copying the relevant folder from **<LR installation directory>\mobileRemoteAgent** to a location of your choice on your server. The following table lists the supported operating systems and their corresponding file directories:

OS	Folder
Windows OS	Win32
Linux	LinuxRH3
Mac	Mac

Once you have copied the folder, you initiate the process by typing a command line argument. For example, if you are recording in a Windows environment the command line string may look something like this:

```
mongoose-2.11.exe -e errorLog.txt -r "C:\Program
Files\HP\LoadRunner\mobileRemoteAgent\win32" -C ".cgi" -p 80
```

Common command arguments and their descriptions:

-c	Identify the remote sniffer agent extensions. Always "cgi"
-p	Identify the port and optionally the ip address that you want the mobile sniffer agent to listen on
-r	Location of the mobile sniffer agent. **Path must be absolute or the process will fail.**
-l	Restricts access to the mobile sniffer agent to a specific client identified by the client's IP
-e	Name of the error log

Note: Do not use –g option. (“digest user/password”) There is a defect with this option.

Once a sniffer agent is installed, you manually start the agent and continue the **Record and Analyze** method from the **Recording Wizard**.

For details, see "Recording Wizard" on page 579

Mongoose Details and MIT License

For more information on using Mongoose, see Mongoose Manual

Mongoose License information

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CGI Configuration File Options

The CGI configuration file is located in **<LR installation directory>\mobileRemoteAgent\<win32/Mac/LinuxRH3>/cgi-bin/mobileCGI.conf**

You can configure the following options:

- **timeout-seconds**

  Default =1800

  The timeout session check stops when the process is sleeping due to traffic inactivity; the timeout session resumes as soon as the adapter is recording events.

- **max-pcap-file-size-kb**

  Default = 102400

  Maximum file size check stops when the process is sleeping due to traffic inactivity; the timeout session resumes as soon as the adapter is recording events.

  Since traffic is first written to a buffer file and flashed to the hard drive, the maximum size may be exceeded.

- **log-file-enable**

  Default = 0

Example of CGI configuration file

timeout-seconds 1800

max-pcap-file-size-kb 102400

log-file-enable 0

Recording traffic on local hosts

In certain circumstances,capturing traffic directly on the application sever may not be practical. In this instance, you can capture traffic to your local computer by connect your mobile device to local

host. Clicking Connect will launch the mobile sniffer agent. Continue the **Record and Analyze**method through the **Recording Wizard.**

Configuring a "hotspot" on your local host

In order to record using a local host, you must first configure your machine as a hotspot. There are two ways to accomplish this, either by adhoc or softapp.

For information on Adhoc click here.

From information on SoftApp click here.

Notes:

- Not all the devices support adhoc. For example, some versions of Andoid don't support it.
- Since our agent is using libpcap (the same package used by Wireshark) not all network configurations will be supported by the mobile sniffer agent.
- In order for a hotspot to work, you first need to configure your firewall to support it.
- The hotspot may be implemented with NAT thus the address of the mobile will be identical to the hotspot machine.

Once hotspot as been configured on your local host, you continue to **Record and Analyze** method using the **Recording Wizard**.

For details, see "Recording Wizard" on page 579

Capture Traffic With an External SnifferTool

Most UNIX operating systems have a built-in version of a capture tool. In addition, there are many downloadable capture tools such as Wireshark/tcpdump. When using external tools, make sure that all packet data is being captured and none of it is being truncated. Certain capture utilities require additional arguments. For example, tcpdump requires the -s 0 argument in order to capture the packets without truncating their data. You can also manually create a capture file using the command line utility.

The VuGen command line utility, lrtcpdump, is located in the product's bin folder. There is a separate utility for each of the platforms:

- lrtcpdump.exe (Windows)
- lrtcpdump.hp9
- lrtcpdump.ibm
- lrtcpdump.Linux
- lrtcpdump.solv4

In addition, there are many downloadable capture tools such as Wireshark/tcpdump. Once the capture file has been created, you can generate the script by using the **Analyze Traffic** method from the **Recording Wizard**.

For details, see "Recording Wizard" on page 579

Analyzing Traffic

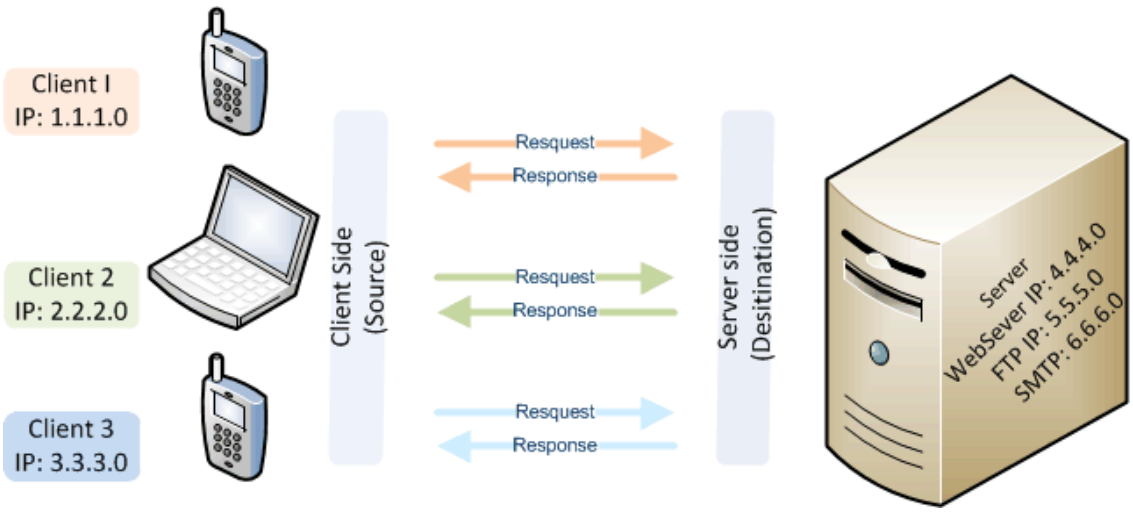
You will need to specify which IP address the Recording Wizard should analyze. The **AnalyzeTraffic** option from the **Recording Wizard** provides two methods to specify which IP address's traffic to include in the script VuGen is generating. As illustrated in the graphic , traffic is identified either from the server side or from the client side. You will need to specify which IP address's traffic, either client or server side, to include in the script.

Server Side (Destination)

You can configure multiple IP addresses on one server. For example you may be running SMTP , FTP and a web server all on the same computer with unique IP addresses. When the sniffer agent creates the capture file it will include all events to all the services on the server. However, by configuring server side traffic, you can specify the IP address and port where your application under test resides. This method is useful when you are certain that only one client is communicating with the server.

Client Side (Source)

You can identify the client that has been communicating with the server as the IP address to analyze. This way, only the events from the client's IP address will be included in the script VuGen will generate.



Filter Options

You can specify additional IP addresses to either include or exclude during code generation **Recording Options > Traffic Analysis > Traffic Filters**. For details, see "Traffic Analysis > Traffic Filter" on page 330.

Recording with Emulation

Testing mobile applications with an emulator is a solid solution for many mobile devices. You can install a third party emulation application on your local computer and record events with the Record Emulator method from the Recording Wizard.

The Record Emulator method requires three settings in order to start the emulator:

- Emulator to record
- Command line
- Working directory

For details, see "Record Emulation Dialog Box" on page 581.

Below is a table including emulator download site and **Record Emulator** settings for Web OS:

Mobile OS	Link to Download SDK	Emulator to record (example)	Command line (example)	Working directory (example)
Web OS	Web OS Emulator Download	C:\Program Files\Palm\SDK\bin\palm-emulator.bat	empty	C:\Program Files\Palm\SDK\bin

Recording with a Mac iPhone Emulator

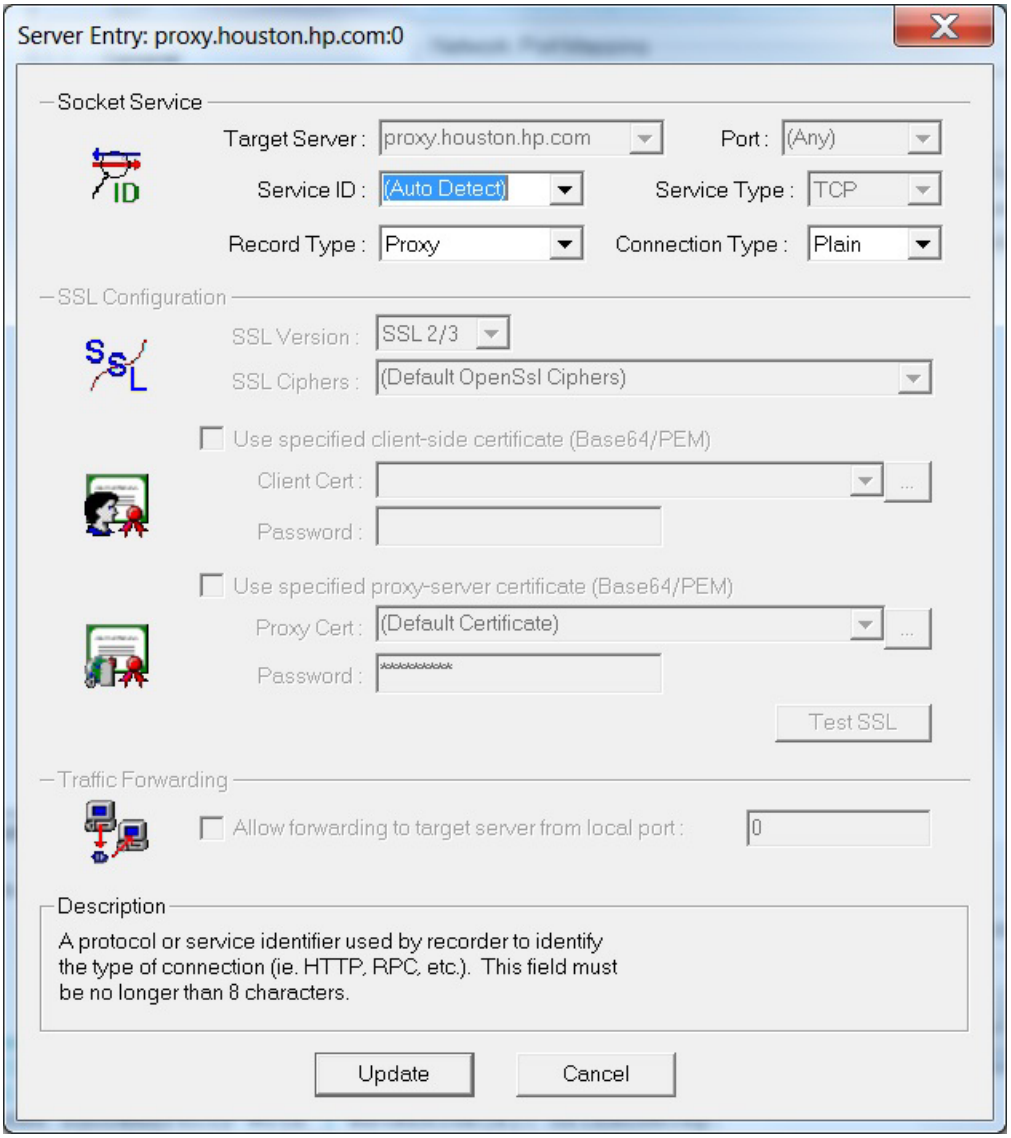
Using the Record and Analyze method from the Recording Wizard, you can record an iPhone script using an emulator.

- Download the iPhone emulator and install your Mac. iPhone Emulator Download.
- Use the Record and Analyze method from the Recording Wizard to create capture file.

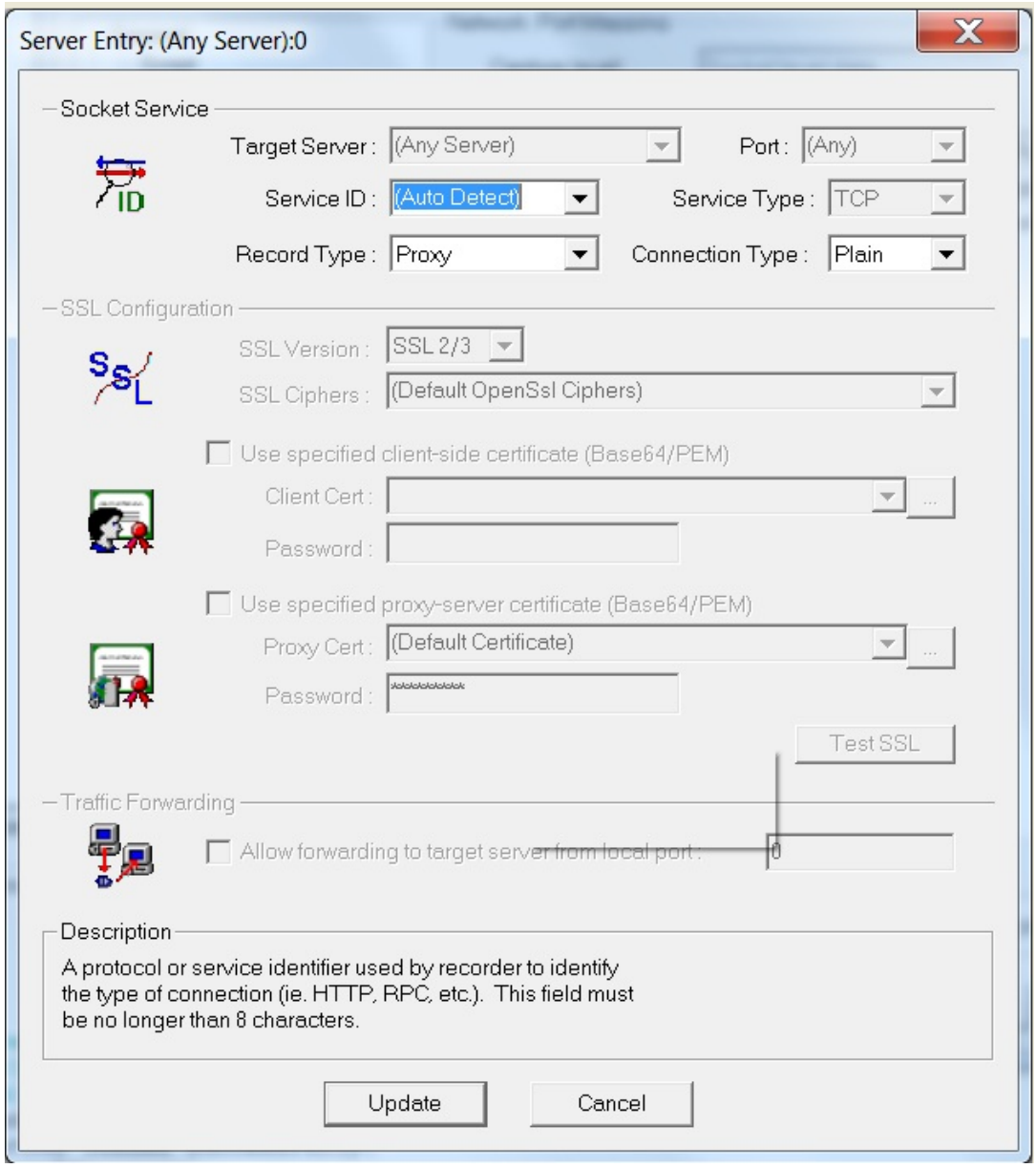
Recording with a Google Android Emulator

- If you are recording with Google Android Emulator Version 2.0 and above, apply the following workaround if you are experiencing problems recording:

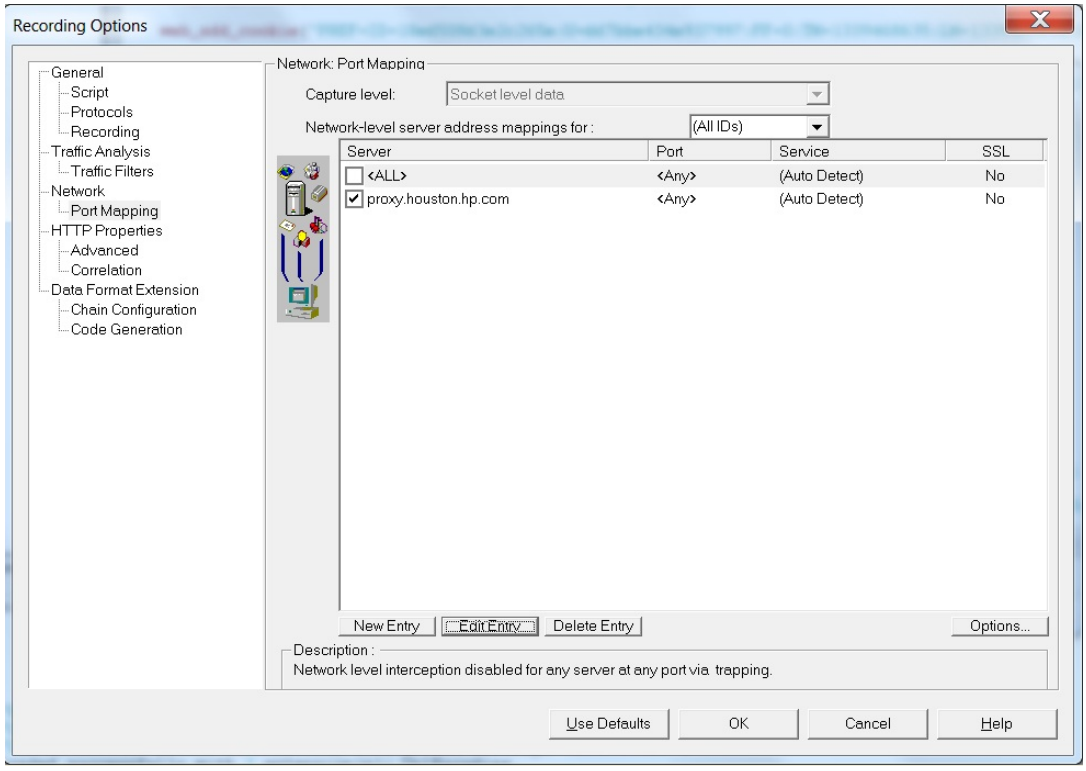
1. Enter a new Port Mapping by selecting **Recording Options > Network > Port Mapping** and select **New Entry.**
2. Specify a Target server and port.



3. Enter a second Port Mapping entry without changing any details.



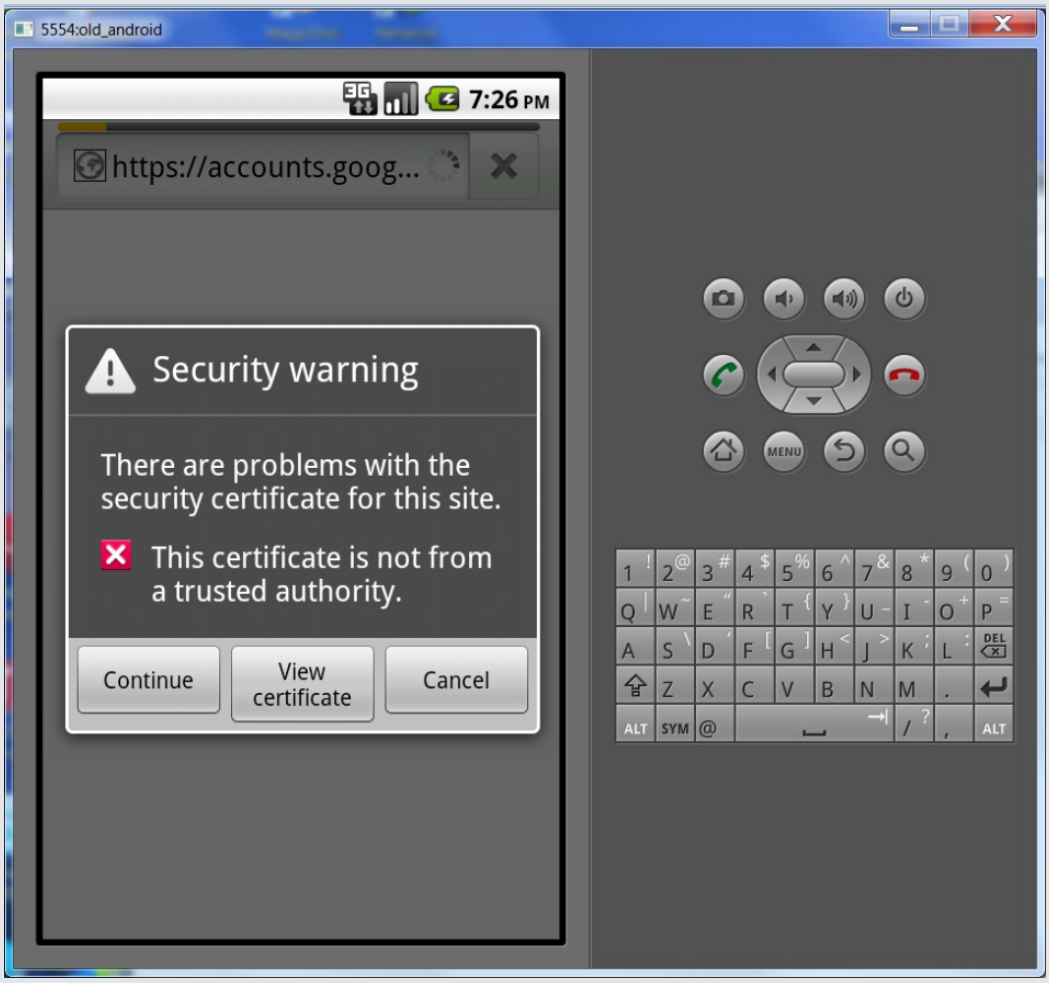
4. Disable the second entry so all traffic is handled by the first entry in the Port Mapping.



Note:

- This limitation was fixed in Google Android Emulator Version 4.03
- While recording a SSL site, you may encounter a warning message stating that there are

problems with the certificate for the site. Click continue to proceed with the recording.



How to Create a Script by Analyzing Traffic

This task describes how to create a script using a network traffic file (capture file).

1. **Create a capture file on a Windows Platform - optional**

   Create a capture file containing a log of all TCP traffic over the network on a Windows platform. Use a downloadable capture tool or use the tool provided in the product's bin folder, lrtcpdump.<platform>.

   a. Run the capture utility in a command window lrtcpdump -f<file_name>.cap. lrtcpdump prompts you to select a network card.

   b. Type in the number of the interface card (if there are multiple ones.) and click Enter.

   c. Perform typical actions within your application.

   d. Return to the command window and click Enter to end the capture session.

e. Place the capture file on the network in a location accessible to the machine running VuGen.

2. **Create a capture file on a UNIX Platform - optional**

Create a capture file containing all TCP traffic over the network on a UNIX platform.

a. Locate the appropriate lrtcpdump utility for your platform in the product's bin folder. Copy it to a folder that is accessible to your UNIX machine. For example, for an HP platform, copy lrtcpdump.hp9. If using FTP, make sure to use the binary transfer mode.

b. Switch to the root user to provide execution permissions: chmod 755 lrtcpdump.<platform>

c. If there are multiple interface cards, lrtcpdump uses the first one in alphabetical order. To get a complete list of the interfaces, use the ifconfig command.

d. Run the utility with its complete syntax, specifying the interface and file name. For example, lrtcpdump.hp9 -ietho -f<file_name>.cap. The capturing of the network traffic begins.

e. Perform typical actions within your application.

f. Return to the window running lrtcpdump and follow the instructions on the screen to end the capture session.

g. Place the capture file on the network in a location accessible to the machine running VuGen.

3. **Start the Recording Wizard**

Click the **Start Record** button. Select the **Analyze Traffic** method.

For details, see"Recording Wizard" on page 579

4. **Specify the location of the Capture File**

Enter or browse to the location of the capture file.

5. **Specify traffic information**

Specify a capture file and the section of the script into which you want to load the traffic: vuser_init, Action, or vuser_end.

Indicate whether you want to analyze server or client side traffic.

For details on analyzing traffic, see "Analyzing Traffic" on page 571.

6. Configure the SSL-optional

Click the SSL Configuration button to add SSL certificates. This is necessary in order to analyze traffic from a secure server.

For details, see "SSL Configuration Dialog Box" on page 710.

Note: If you have load on the server that prevents getting responses on time, you can create a script based on requests only by editing the registry.

Go to regedit and add a DWORD key to the below location:

```
Location:
```

```
Software\\Mercury
Interactive\\LoadRunner\\Protocols\\HTTP\\Analyzer\

Key:

AllowAutomaticOutApiEvents = 1
```

How to Record and Analyze a Script for Mobile Applications

This task describes how to record and analyze a script with the Recording Wizard

1. **Prerequisites**

   If you are recording the capture file on a remote server, you must manually start the mobile sniffer agent.

   If you are recording the capture file on the local host, you must connect your mobile device to the local host.

   For details, see "Recording Traffic into a Capture (Sniffer) File" on page 566.

2. **Create new Mobile App (HTTP/HTML) script**

   Click the **Start Record** button.

   Select the **Record and Analyze** method.

   For details, see "Recording Wizard" on next page

3. **Specify mobile sniffer agent location**

   Enter the URL where the mobile sniffer agent resides. For example, if the mobile sniffer agent is on the local machine enter http:\\localhost.

4. **Configure mobile sniffer agent**

   Specify the Record network interface. This is the network adapter to which you want the mobile sniffer agent to listen.

   Specify which section of the script into which you want to load the traffic: vuser_init, Action, or vuser_end.

5. **Select Start Recording**

   Record the desired business process.

6. **Select Stop Recording**

   VuGen will generate the capture file.

   Save capture file on the local machine when you are prompted.

7. **Analyze Traffic**

   For details, see "How to Create a Script by Analyzing Traffic" on page 576 starting with Step 4.

Recording Wizard

This wizard enables you to configure record and analyze setting for the Mobile App (HTTP/HTML) protocol.

To access	Click the **Start Record** button
Important information	"Recording Traffic into a Capture (Sniffer) File" on page 566 "Analyzing Traffic" on page 571
Wizard map: Analyze Traffic	The following is the wizard map if you select Analyze Traffic in the Recording Method page: "Recording Method Dialog Box" below > "Analyze Traffic Dialog Box" on next page
Wizard map: Record and Analyze Traffic	The following is the wizard map if you select Record and Analyze in the Recording Method page: "Recording Method Dialog Box" below >"Configure and Record Dialog Box" on next page >"Analyze Traffic Dialog Box" on next page
Wizard map: Record Emulator	The following is the wizard map if you select Record Emulator in the Recording Method page: "Recording Method Dialog Box" below > "Record Emulation Dialog Box" on page 581
Relevant tasks	"How to Create a Script by Analyzing Traffic" on page 576 "How to Record and Analyze a Script for Mobile Applications" on previous page

Recording Method Dialog Box

This dialog box enables you to select a recording method

To access	**VuGen > Start Record** button
Relevant tasks	"How to Create a Script by Analyzing Traffic" on page 576 "How to Record and Analyze a Script for Mobile Applications" on previous page

User interface elements are described below:

UI Element	Description
Analyze Traffic	Create script by converting an existing capture file . For details, see "Recording Traffic into a Capture (Sniffer) File" on page 566
Record and Analyze Traffic	Create script by recording server traffic and converting the resulting capture file.
Record Emulator	Create script by recording events with a device emulator installed on your local machine running VuGen.

Analyze Traffic Dialog Box

This dialog box enables you to configure settings for script generation from an existing capture file.

To access	**VuGen > Recording Method > Next**
Relevant tasks	Add list of cross references using bullets for multiple entries. Do not use periods.

User interface elements are described below:

UI Element	Description
Capture File	The name of a capture file containing the server traffic, usually with a cap extension.
	**Browse.** Allows you to select a capture file to import.
Filter based on server side	The IP address and port of the server whose traffic you want to examine.
Filter based on client side	The IP address of the client whose traffic you want to examine.
Record into Action	The section into which you want to record: vuser_init, Action, or vuser_end. For actions you want to repeat, use the Action section. For initialization steps, use vuser_init.
Filter Options	Opens the Traffic Filters node in the Recording Options dialog box. This allows you to specify which IP addresses to include or exclude from the script during script generation.
SSL Configuration	Opens the SSL Configuration Dialog Box which allows you to add SSL certificates to analyze traffic from a secure server.

Configure and Record Dialog Box

This dialog box enables you to configure settings to record traffic to a pcap file.

To access	Recording Wizard > Recording Method > Next
Relevant tasks	"How to Create a Script by Analyzing Traffic" on page 576 "How to Record and Analyze a Script for Mobile Applications" on page 578

User interface elements are described below:

UI Element	Description
Mobile sniffer agent location	**Host URL.** URL where the mobile sniffer agent resides. **Port.** Specify the port for the mobile sniffer agent to listen.
Mobile sniffer configuration:	**Record Network Interface.** Servers and computers can have multiple network adapters. The setting enables you to select the network adapter you want the mobile agent to listen on. **Record into Action.** The section into which you want to record: vuser_init, Action, or vuser_end. For actions you want to repeat, use the Action section. For initialization steps, use vuser_init.
Recording	**Start Recording.** Begin recording traffic. **Stop Recording: Stop recording traffic.**

Record Emulation Dialog Box

This dialog box enables you to configure the settings to use a mobile emulator to record a script.

To access	Recording Wizard > Recording method > Next

User interface elements are described below:

UI Element	Description
Emulator to record	Specify the location of the emulator application executable.
Command line	Specify the arguments that the emulator application requires to run the emulator.
Working directory	Specify the location of the of the emulator application.
Action to record into	Specify the section into which you want to record: vuser_init, Action, or vuser_end. For actions you want to repeat, use the Action section. For initialization steps, use vuser_init.
Options button	Access the **Recording Options** pane.

Mobile Ajax TruClient Protocol

Based on LoadRunner's innovative Ajax TruClient technology, Mobile TruClient enables you to test web applications designed for mobile devices.

With this protocol you can:

- Simulate various mobile browsers.
- Develop scripts that are recorded on the user level making them clear and easily maintained.

The following flow chart illustrates the workflow for using the Mobile TruClient protocol:

How to Record a Script with Mobile TruClient

1. Create new Mobile TruClient script
2. **Select Start Script Development**

   Define mobile device, user agent and display size in the Mobile Settings Dialog Box which opens automatically.

   For details, "Mobile TruClient Device Manager Dialog Box" on next page.
3. **Record business process**

   For details on using TruClient's functionality, see "Ajax TruClient Protocol" on page 396.

How to Add, Remove, and Import Mobile Device Settings for Mobile TruClient

How to Create a Custom Device Using the Mobile Device Manager

The Mobile TruClient device manager is delivered with the settings for many popular mobile devices, however, you can easily add a custom device.

1. Click the Mobile TruClientDevice Manager from the VuGen toolbar.
2. Select **Add Mobile Device**. This opens the **Add Mobile Device** dialog box.
3. Enter the name of the device you would like to add or select an an existing device from the drop-down list to customize.
4. Specify the User Agent. For details, see "Mobile TruClient Device Manager Dialog Box" on next page.
5. Specify the Display. For details, see "Mobile TruClient Device Manager Dialog Box" below.

How to Remove a Mobile Device

1. Click the Mobile TruClientDevice Manager from the VuGen toolbar.
2. Select Remove Mobile Device. This opens the Remove Mobile Devicedialog box.
3. Highlight the unwanted device and click remove.

How to Import a Mobile Device Settings to Your Script

The import feature can be used to import mobile devices created in other users' scripts.

1. Open a script that contains custom device settings.
2. Click the Mobile TruClientDevice Manager from the VuGen toolbar.
3. Select the **Import Device Settings**. This opens the **Import Device Settings** dialog box.
4. Highlight the custom device and select import.

Mobile TruClient Device Manager Dialog Box

This utility enables you to configure, add and import mobile device settings that are used with the Mobile TruClient Protocol.

To access	Use one of the following: • **Script development** button  • **Tools > Mobile TruClient Device Manager** • **Mobile Settings** button 
Relevant tasks	"How to Add, Remove, and Import Mobile Device Settings for Mobile TruClient" on previous page "How to Record a Script with Mobile TruClient" on previous page

UI Element	Description
Mobile Device	Select the mobile device type you want to test.
User Agent	Specify the header string that is sent to server to identify your mobile device. Once you have a selected a device, the default header value will appear. However, this header string can be modified.
Display	Specify the height and width of your mobile device screen. Mobile TruClient will open browser window according to the display settings.

.NET Protocol

.NET Protocol Overview

Microsoft .NET Framework provides a foundation for developers to build various types of applications such as ASP.NET, Windows Forms, Web Services, distributed applications, and applications that combine several of these models.

VuGen supports .NET as an application level protocol. VuGen allows you to create Vuser scripts that emulate users of Microsoft .NET client applications created in its .NET Framework. VuGen records all of the client actions through methods and classes, and creates Vuser scripts in C Sharp or VB .NET.

By default, the VuGen environment is configured for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation) applications. Contact Customer Support for information on how to configure VuGen to record applications created with other client-server activity.

For more information about .NET and the above environments, see the MSDN Web site, http://msdn2.microsoft.com.

Viewing Data Sets and Grids

When you record a method returning a dataset, data table, or data reader action, VuGen generates a grid for displaying the data.

When working with a data reader, VuGen collects the data retrieved from each **Read** operation and converts it to the replay helper function, **DoDataRead**.

For example, after recording the following application code,

```
SqlDataReader reader = command.ExecuteReader();
     while( reader.Read() )
       {
           // read the values, e.g., get the string located in column
1
           string str = reader.GetString(1)
       }
```

VuGen generates the following lines in the script:

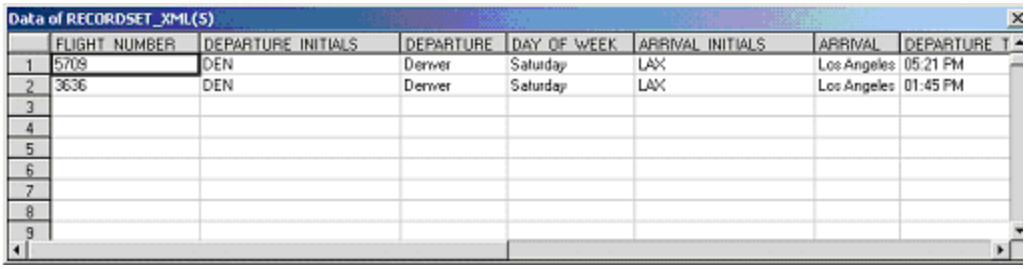
```
SqlDataReader_1 = SqlCommand_1.ExecuteReader();
LrReplayUtils.DoDataRead(SqlDataReader_1, out valueTable_1, true, 27);
```

where the two parameters indicate that during recording, the Application read all 27 available records. Therefore, during replay the script will read all available records.

In addition, VuGen generates a data grid containing all the information retrieved by the **Read** operations.

During replay you can use the output data table, containing the actual retrieved values, for correlation and verification. For more information regarding the **DoDataRead** function, see the Function Reference (**Help > Function Reference**).

When applicable, VuGen displays grid steps in the Step Navigator, and displays the associated grids in the Snapshot pane.



The dataset is stored in an XML file. You can view this XML file in the script's data/datasets folder. The data files are represented by an <*index_name*>.xml file, such as 20.xml. Since one file may contain several data tables, see the file **datasets.grd**, which maps the script index to the file index to determine which XML contains the data.

Recording WCF Duplex Communication

WCF (Windows Communication Foundation) is a programming model that unifies Web Services, .NET Remoting, Distributed Transactions, and Message Queues into a single Service-oriented programming model for distributed computing.

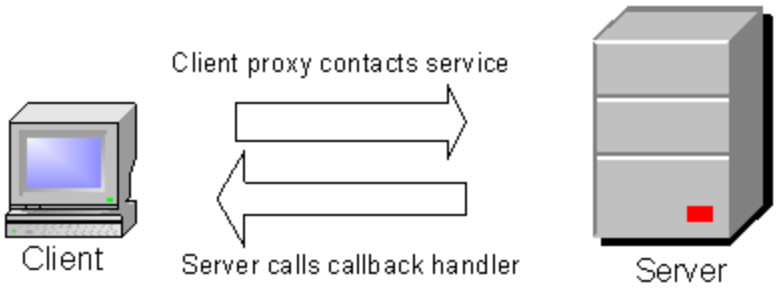
WCF creates a proxy object to provide data for the service. It also marshalls the data returned by the service into the form expected by the caller.

In addition to general support for the WCF environment, VuGen provides specialized support for applications that use WCF's duplex communication. In duplex communication, the client proxy contacts the service, and the service invokes the callback handler on the client machine. The callback handler implements a callback interface defined by the server. The server does not have to respond in a synchronous manner—it independently determines when to respond and invoke the callback handler.

Communication Between Client and Server

The communication between the client and server is as follows:

- The server defines the service contract and an interface for the callback.
- The client implements the callback interface defined by the server.
- The server calls the callback handler in the client whenever needed.



When trying to record and replay duplex communication, you may encounter problems when the script calls the original callback methods. By default, the callback handlers are not included in the filter. You could customize the filter to include those callback handlers. However, the standard playback would be ineffective for a load test, since many of the callbacks are local operations such

as GUI updates. For an effective load test you cannot replay the original callback method invoked by the server.

VuGen's solution is based on replacing the original callback handler with a dummy implementation. This implementation performs a typical set of actions that you can customize further for your application.

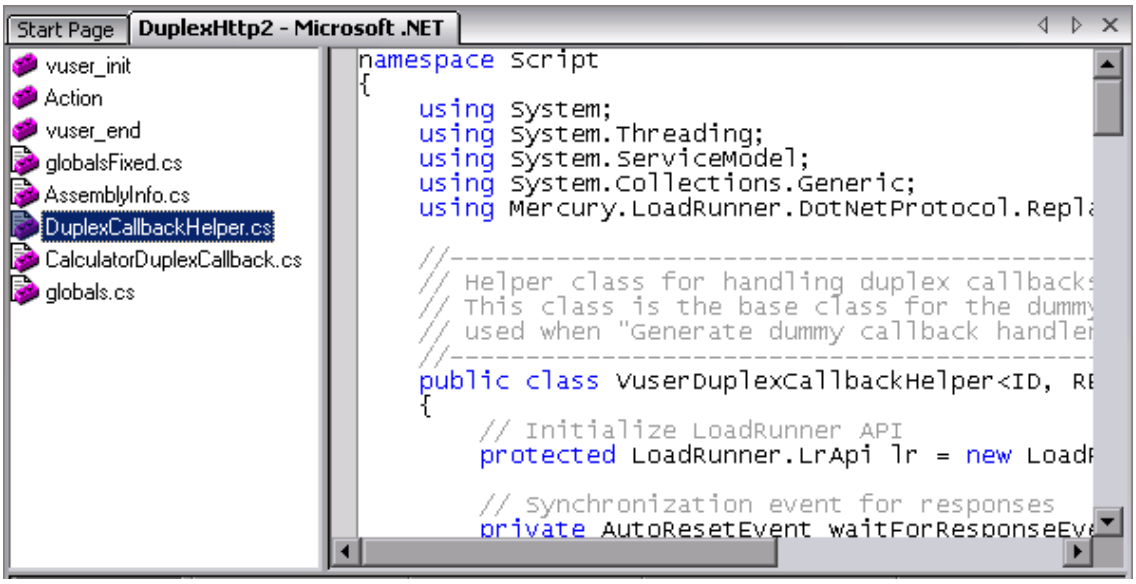
You instruct VuGen to replace the original callbacks by activating the **Generate Dummy Callback Handler** recording option. For more information, see "Remote Objects Property" on page 310.

VuGen Implementation of a Duplex Callback

As part of the duplex communication solution, VuGen generates two support files:

- `DuplexCallbackHelper.<language>`
- *Callback_Name*`.<language>`

The following example shows the generated files for a Calculator application using duplex communication:



The Helper file serves as a general template for working with duplex callback handlers. It serves as a base class for the implementation of the callback.

The second file, **Callback_Name**, contains the implementation of the callback. The name of the callback implementation class is **Vuser**<*xxxx*> where *xxxx* is the name of the callback interface and it inherits from the **VuserDuplexCallbackHelper** class defined in the Helper file. VuGen creates separate implementation files for each interface.

This file performs two primary tasks:

Set Response. It stores the data that came from the server in a map. It stores them with sequential IDs facilitating their retrieval. This method is called from the implementation of the callback interface. The following sample code demonstrates the dummy implementation of a callback method named **Result**. The method's arguments are stored in the map as an object array.

```
//  ------------------------------------------------------------------
--------
        public virtual void Result(string operation, double result) {
            // Add here your own callback implementation and set the
response data
            SetResponse(responseIndex++, new object[] {
                        operation,
                        result});
}
```

- **Get Response.** Waits for the next response to arrive. The implementation of GetNextResponse retrieves the next response stored in the map using a sequential indexer, or waits until the next response arrives.

The script calls GetNextResponse at the point that the original callback handler was called during recording. At that point, the script prints a warning:

```
// Wait here for the next response.
// The original callback during record was:
```

Replacement of the Callback in the Script

When you enable the **Dummy Callback** option (enabled by default), VuGen replaces the original duplex callback handlers with dummy implementations. The dummy implementation is called Vuser <*Callback Name*>. At the point of the original callback handler, the script prints a warning indicating that it was replaced.

Customizing the Dummy Implementation

You can modify the implementation file to reflect your environment. This section contains several suggested customizations.

Timeouts

The default timeout for which the callback waits for the next response is 60000 msec, or one minute. To use a specific timeout, replace the call to **GetNextResponse** with the overloading method which gets the timeout as an argument as shown below. This method is implemented in the callback implementation file <*Callback_Name*> listed in the left pane after the **DuplexCallbackHelper** file.

```
// Get the next response.
        // This method waits until receiving the response from the
server
        // or when the specified timeout is exceeded.
        public virtual object GetNextResponse(int millisecondsTimeout)
{
            return base.GetResponse(requestIndex++,
millisecondsTimeout);
        }
```

To change the default threshold for all callbacks, modify the **DuplexCallbackHelper** file.

```
 // Default timeout threshold while waiting for response
        protected int millisecondsTimeoutThreshold = 60000;
```

Key Identifier

Many applications assign key identifiers to the data, which connects the request and response to one another. This allows you to retrieve the data from the map using its ID instead of the built-in incremental index. To use a key identifier instead of the index, modify the file <*Callback_Name*> replacing the first base template parameter, **named ID**, with the type of your key identifier. For example, if your key identifier is a string you may change the first template argument from **int** to **string**:

```
public class VuserXXX : VuserDuplexCallbackHelper<string, object>
```

In addition, you may remove the implementation of GetNextResponse() and replace it with calls to GetResponse(ID) defined in the base class.

Return Values

By default, since VuGen supports *OneWay* communication, the implementation callback does not return any value or update an output parameter when it is invoked.

```
 public virtual void Result(string operation, double result) {
            // Add here your own callback implementation and set the
response data
```

If your application requires that the callback return a value, insert your implementation at that point.

Get Response Order

In VuGen's implementation, a blocking method waits for each response. This reflects the order of events as they occurred during recording—the server responded with data. You can modify this behavior to execute without waiting for a response or to implement the blocking only after the completion of the business process.

Find Port

The **FindPort** method in the Helper file is a useful utility that can be used in a variety of implementations. The Helper class uses this method to find unique ports for running multiple instances of the script. You can utilize this utility method for other custom implementations.

Recording Server Hosted By Client Applications

If the communication in your system is a server hosted by a client, VuGen's default solution for duplex communication will not be effective. In server hosted by client environments, it is not true duplex communication since the client opens the service and does not communicate through the Framework. For example, in queuing, the client sends a message to the service and opens a response queue to gather the responses.

To emulate a server hosted by a client, use the pattern depicted in the above solution—replace the original response queues with dummy callbacks and perform synchronization as required. For more information, contact HP support.

Asynchronous Calls

When VuGen records asynchronous calls on remote objects, you can specify how the calls are handled in the ".NET > Recording - Recording Node" on page 308. These options are particularly relevant for .NET Remoting and WCF environments.

You can configure VuGen to one of the following options:

- **Call original callbacks by default.** Uses the recorded application's original callback when generating and replaying the script. If the callback method is explicitly excluded by a filter, the callback will be excluded even if you enable this option. If your callbacks perform actions that are not directly related to the business process, such as updating the GUI, then make sure to disable this option.
- **Generate asynchronous callbacks.** This option defines how VuGen will handle callbacks when the original callbacks are not recorded. This is relevant when the above option, **Call original callbacks** is disabled or when the callbacks are explicitly excluded.

  When you enable this option, it creates a dummy method which will be called during replay instead of the original callback. This dummy callback will be generated in the **callbacks.cs** section of the script.

  When you disable this option, VuGen inserts a NULL value for the callback and records the events as they occur.

The following segment shows script generation for a Calculator client, when **Generate asynchronous callbacks** is enabled.

To display the callback method, OnComplete1, you click on the **callback.cs** file in the left pane.

The following segment shows script generation when the option is disabled. VuGen generates a NULL in place of the callback and records the events of the callback as they occur.

Note: If you recorded a script with specific recording options, and you want to modify them, you do not need to re-record the script. Instead you can regenerate the script with the new settings.

For more information, see ".NET > Recording - Recording Node" on page 308.

Recording Dual HTTP Bindings

If your application employs dual HTTP Binding, since HTTP is inherently not a duplex protocol, the framework uses a standard port to receive response data being passed to the callback. When you attempt to run multiple instances of your application, you may be unable to do so using the same port number. VuGen provides you with an option of replacing the original client base address's port number with a unique port.

When you enable the **Generate Unique Client Base Address** recording option, VuGen checks the type of communication used by the application. If it detects dual HTTP communication, **WSDualHttpBinding**, it runs the **FindPort** utility (provided in LrReplayUtils) in the Helper file and finds unique ports for each instance of the callback.

This option is enabled by default. It is only relevant when you enable the above option, **Generate dummy callback handler**.

When you enable this option, VuGen generates the following code in the script:

```
#warning:  Code Generation Warning
    // Override the original client base address with a unique port
number
    DualProxyHelper.SetUniqueClientBaseAddress<XXXX>(YYYYY);
```

For more information, see ".NET > Recording - Recording Node" on page 308.

Connection Pooling

ADO.NET providers deploy a feature called **connection pooling** which can significantly influence load test accuracy. Whenever only one app domain is used for all Vusers, connection pooling is turned on—.NET Framework keeps the database connections open and tries to reuse them when a new connection is requested. Since many Vusers are executed in the context of a single application domain, they may interfere with one another. Their behavior will not be linear and that may decrease their accuracy.

In the .NET run-time settings, the AppDomain Per Vuser property enables execution of each Vuser in a separate app domain (true by default). This means that there is connection pooling in the scope of each Vuser, but the Vusers will not interfere with one another. This setting provides more accuracy, but lower scalability.

If you disable this option, you need to manually disable connection pooling for the database.

The following table describes how to manually disable connection pooling:

Provider	Option
.NET Framework Data Provider for SQL Server	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for Oracle	"Pooling=false" or "Pooling=no"
.NET Framework Data Provider for ODBC	Connection pooling is managed by an ODBC Driver Manager. To enable or disable connection pooling, use the ODBC Data Source Administrator (found in Control Panel or the Administrative Tools folder). The Connection Pooling tab allows you to specify connection pooling parameters for each of the installed ODBC drivers.
.NET Framework Data Provider for OLE DB	"OLE DB Services=-2"
Oracle Data Provider for .NET	"Pooling=false"

Adaptive Server Enterprise ADO.NET Data Provider	"Pooling=False"

Debugging .NET Vuser Scripts

You can compile a .NET Vuser script to check its syntax, without running the script. To compile the script directly from VuGen, press Shift+F5 or select **Vuser > Compile**. If VuGen detects a compilation error, it displays the error in the Output window. Double-click on the error to go to the problematic line in the script.

To run the script directly from VuGen, press F5 or select **Replay > Run**. Breakpoints and step-by-step replay are not supported in VuGen's editor window for .NET Vusers. To debug a script and run it with breakpoints or step-by-step, run it from within Visual Studio .NET as described below.

Viewing Scripts in Visual Studio

Visual Studio provides you with additional tools to view, edit, and debug your Vuser scripts. You can add breakpoints, view variable values, add assembly references, and edit the script using Visual Studio's IntelliSense. You can also run the script in a step-by-step mode for debugging.

When you save your script, VuGen creates a Visual Studio 2010 solution file, **Script.sln**, in your script's folder. You can open the solution file in Visual Studio .NET and view all of its components in the Solution Explorer.

To open the Vuser script in Visual Studio 2010, select **Design > Open Script in Visual Studio** or click the **Visual Studio** button  on the VuGen toolbar.

> Note that if the Vuser script was created with a version of LoadRunner 11 or older, LoadRunner will open the script in Visual Studio 2005.

Double-click the appropriate section in the Solution Explorer, such as **vuser_init.cs**, to view the contents of the script.

> Note that VuGen automatically loads all of the necessary references that were required during recording. You can add additional references for use during compilation and replay through the Solution Explorer. Select the **Reference** node and select **Add Reference** from the right-click menu.

Click on **globals.cs** or **globals.vb** in the Solution Explorer to view a list of the variables defined and used by your script.

.NET Filters Overview

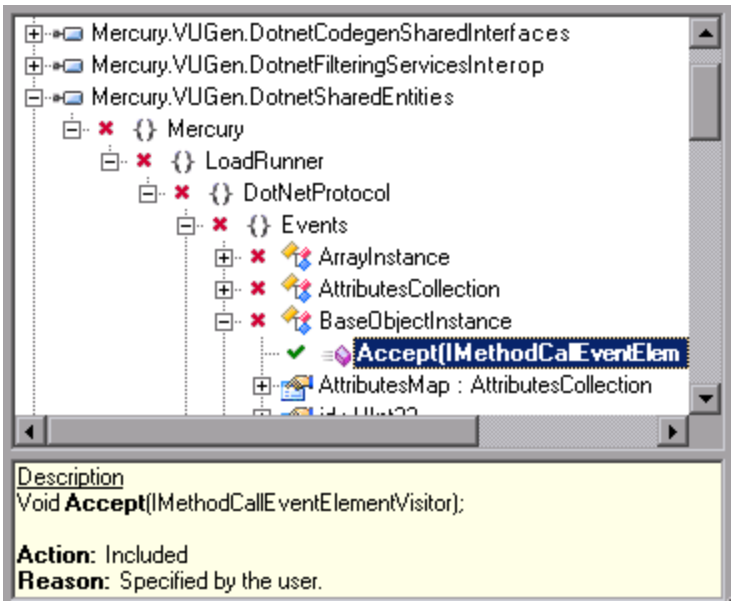
Recording filters indicate which assemblies, interfaces, namespaces, classes, or methods to include or exclude during the recording and script generation.

By default, VuGen provides built-in system filters for .NET Remoting, ADO.NET, Enterprise Services, and WCF (Windows Communication Foundation). These filters were designed to include the relevant interfaces for standard ADO.NET, Remoting, Enterprise Services, and WCF. VuGen also allows you to design custom filters.

Custom filters provide several benefits:

- **Remoting.** When working with .NET Remoting, it is important to include certain classes that allow you to record the arguments passed to the remote method.
- **Missing Objects.** If your recorded script did not record a specific object within your application, you can use a filter to include the missing interface, class or method.
- **Debugging.** If you receive an error, but you are unsure of it's origin, you can use filters to exclude methods, classes, or interfaces in order to pin-point the problematic operation.
- **Maintainability.** You can record script in higher level, make script more easy to maintain and to correlate.

A filter manager lets you manipulate existing custom filters. It displays the assemblies, namespaces, classes, methods, and properties in a color-coded tree hierarchy.



The bottom pane provides a description of the assembly, namespace, class, method, property, or event. It also indicates whether or not it is included or excluded and a reason for the inclusion or exclusion.

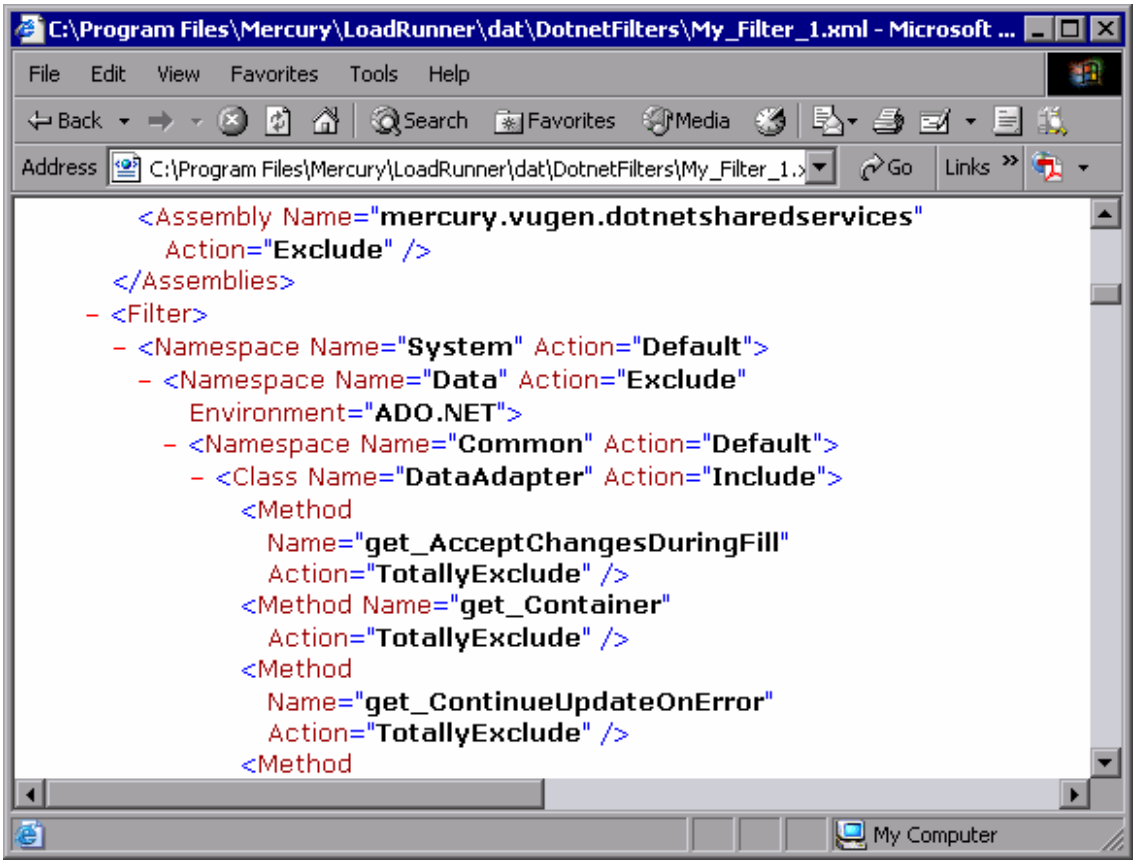
.NET Filters - Advanced

In the Filter Manager's tree hierarchy, it only displays public classes and methods. It does not show non-public classes or delegates.

You can add classes or methods that are not public by manually entering them in the filter's definition file.

The filter definition files, **<filter_name>.xml** reside in the dat\DotnetFilters folder of your installation. The available Action properties for each element are: **Include**, **Exclude**, or **Totally Exclude**. For more information, see "Filter Manager [.NET Protocol]" on page 304.

By default, when you exclude a **class**, the Filter Manager applies **Exclude**, excluding the class, but including activity generated by the excluded class. When you exclude a **method**, however, it applies **Totally Exclude**, excluding all referenced methods.



For example, suppose Function A calls function B. If Function A is **Excluded**, then when the service calls Function A, the script will include a call to Function B. However, if function A is **Totally Excluded**, the script will not include a call to Function B. Function B would only be recorded if called directly—not through Function A.

VuGen saves a backup copy of the filter as it was configured during the recording, **RecordingFilterFile.xml**, in the script's **data** folder. This is useful if you made changes to the filter since your last recording and you need to reconstruct the environment.

Guidelines for Setting .NET Filters

When testing your .NET application, your goal is to determine how the server reacts to requests from the client. When load testing, you want to see how the server responds to a load of multiple users.

When recording a .NET application, your script may include calls to methods that do not affect the server, such as calls to a local utility or the GUI interface. These calls are usually not relevant to your testing goals, and it would be correct to filter them out.

The built-in filters, .NET Remoting, ADO.NET, Enterprise Services, and WCF, were designed to record only the server related traffic relevant to your testing goals. In some instances, however, you may need to customize filters to capture your .NET application's calls or exclude unnecessary calls. Using the Filter Manager, you can design custom filters to exclude the irrelevant calls and capture the server related calls.

Before creating a test, we recommend that you become familiar with your application and determine its primary classes and methods, so that you will know which ones to include in your recording.

If you are not familiar with your application's classes, you can use **Visual Studio** or a **Stack Trace** to help you determine which methods are being called by your application in order to include them in the filter. VuGen allows you to record with a stack trace that logs all of the methods that were called by your application.

Once you determine the required methods and classes, you include them using the Filter Manager. When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Tip: Strive to modify the filter so that your script will compile (Shift+F5) inside VuGen—a test with correct syntax. Then customize the filter further to create a functional script that runs inside VuGen.

Note that if you plan to add manual code to your script such as control flow or message statements, make sure to do so after you have a functional script that runs inside VuGen. The reason for this is that if you re-record a script or regenerate the script, you will lose all of the manual changes.

Determining which Elements to Include or Exclude

When designing a custom filter, we recommend that you start by choosing the appropriate built-in filter as a base filter. You can then customize the filter using one of the following approaches:

- **Top Down Approach.** An approach in which you include the relevant namespace and exclude specific classes that are not part of the client-server activity. This is recommended if you are familiar with your application and you can identify a well-defined assembly which implements all client-server activity without involving any GUI elements, such as MyDataAccessLayer.dll.
- **Bottom up Approach.** An approach in which you use the default filter and refine it by adding individual methods or classes. Use this approach if you cannot identify a well-defined layer or if you are not familiar with your application. Do not add all AUT assemblies and then try to remove extra component one by one.

The following section provides guidelines on when to include or exclude elements.

- If, as a result of your including a class, your script has many unrelated method calls, try modifying the filter to exclude the irrelevant methods.
- If you identify a non-client/server call in your script, exclude its method in the filter.
- During recording, VuGen may detect an unknown input argument, for example, an argument whose construction it had never encountered before. If this argument supports serialization, VuGen **serializes** it by saving it to a file in a special format. During replay, VuGen reconstructs the argument by **deserializing** it.
- VuGen serializes objects passed as arguments that were not included by the filter. We

recommend that you include this object in the filter in order to track its construction and activity instead of using it in its serialized form. You can identify serialized objects in the script by searching for calls to the **LrReplayUtils.GetSerializedObject** method or, in WCF environments, **LrReplayUtils.GetSerializedDataContract**. VuGen stores serialized objects in the script's **\data\SerializedObjects** folder as XML files with indexes: **Serialization_1.xml**, **Serialization_2.xml** and so forth.

- When no rules are specified for a method, it is excluded by default. However, when the remoting environment is enabled, all remote calls are included by default, even if they are not explicitly included. To change the default behavior, you can add a custom rule to exclude specific calls which are targeted to the remote server.
- Arguments passed in remoting calls whose types are not included by the filter, are handled by the serialization mechanism. To prevent the arguments from being serialized, you can explicitly include such types in order to record the construction and the activity of the arguments.
- Exclude all activity which involves GUI elements.
- Add assemblies for utilities that may be required for the script to be compiled.

For information on how to include and exclude elements, see "Filter Manager [.NET Protocol]" on page 304.

Defining an Effective Filter

When preparing a script, you may need to customize the filter several times in order to achieve the optimal filter. An optimal filter records the relevant methods without introducing a large number of irrelevant calls to the script.

Define an Effective Filter

1. Create a new filter based on one of the built-in filters. If you know that the AUT (Application Under Test) does not use ADO.NET, Remoting, WCF, or Enterprise Services, clear that option since unnecessary filters may slow down the recording.
2. Set the **Stack Trace** option to true for both recording and code generation. Open the Recording Options (ctrl+f7) and select the **Recording** node. Enable **Debug Options: Stack Trace** and **Code Generation: Show Stack Trace**.
3. Record your application. Click **Start Record** (ctrl + r) to begin and Stop (ctrl + f5) to end.
4. View the script's steps. If you can determine the business logic from the steps and apply correlation, you may not need to create custom filters. If, however, the script is very long or hard to maintain or correlate, you should customize the script's filter.
5. Try to identify the high-level method in the call that captures or wraps one or more client server calls. You can do this by opening the AUT source files (if they are available) in Visual Studio or by viewing a Stack Trace of the script.
6. Set the filter to include the relevant methods—you may need to add their assembly beforehand. For tips about including and excluding elements in the filter, see "Guidelines for Setting .NET Filters" on page 593.
7. Record the application again. You should always re-record the application after modifying the filter.
8. Repeat steps 4 through 7 until you get a simple script which can be maintained and correlated.

9. After creating an optimal script, turn off the **Stack Trace** options and regenerate the script. Open the Recording Options (ctrl+f7) and select the **Recording** node. Disable **Debug Options: Stack Trace** and **Code Generation: Show Stack Trace**. This will improve the performance of subsequent recordings.
10. Correlate the script. In order for your test to run properly, you may need to insert a correlation to capture a value and use it at a later point in the script. For more information, see "How to Correlate Scripts - Microsoft .NET" on page 153.

How to Configure Application Security and Permissions

A Security Exception that occurs while recording an application is usually due to a lack of permissions—the recording machine does not have sufficient permissions to record the application. This is common where your application is not local, but on the Intranet or network.

To solve this problem, you need to allow the recording machine to access the application and the script with Full Trust.

One solution is to copy the application and save your script locally, since by default, users have Full Trust permissions to all local applications and folders.

An additional solution is to create new code groups that gives Full Trust to each application folder, and the script folder.

Grant Full Trust permissions to a Specific Folder (Visual Studio NOT installed

1. From the command prompt, run the caspol.exe application.
2. Set the desired permission.

Grant Full Trust Permissions to a Specific Folder (Visual Studio installed)

1. Open the .NET Configuration settings. Select **Start > All Programs > Administrative Tools > Microsoft .NET Framework 2.0 Configuration**. The .NET Configuration window opens.
2. Expand the **Runtime Security Policy** node to show the Code Groups of the machine.
3. Select the **All_Code** node.
4. Select **Action > New**. The Create New Code Group dialog box opens.
5. Enter a name for a new Code Group for your application or script. Click **Next**.
6. Select the **URL** condition type. In the URL box, specify the full path of the application or script in the format `file://...` and click **Next**.
7. Select the **FullTrust** permission set. Click **Next**.
8. Click **Finish** in the Completing the Wizard dialog box. The configuration tool adds your Code Group to the list of existing groups.
9. Repeat the above procedure for all .NET applications that you plan to record.
10. Repeat the above procedure for the Vuser script folder.

Note: Make sure that the script folder has **FullTrust** permissions on all Load Generator

machines that are participating in the test (LoadRunner only).

.NET - Troubleshooting and Limitations

This section describes troubleshooting and limitations for .NET Vuser scripts.

Limitations

The following limitations apply to the VuGen recording of a Microsoft .NET application:

- .NET Vuser scripts support only single-protocol recording in VuGen.
- Direct access to public fields is not supported—the AUT must access fields through methods or properties.
- VuGen does not record static fields in the applications—it only records methods within classes.
- Multi-threaded support is dependent on the client application. If the recorded application supports multi-threading, then the Vuser script will also support multi-threading.
- In certain cases, you may be unable to run multiple iterations without modifying the script. Objects that are already initialized from a previous iteration, cannot be reinitialized. Therefore, to run multiple iterations, make sure to close all of the open connections or remoting channels at the end of each iteration.
- Recording is not supported for Enterprise Services communication based on MSMQ and Enterprise Services hosted in IIS.
- VuGen partially supports the recording of WCF services hosted by the client application.
- Recording is not supported for Remoting calls using a custom proxy.
- Recording is not supported for **ExtendedProperties** property of ADO.NET objects, when using the default ADO.NET filter.
- Applications created with .NET Framework 1.1 which are not compatible with Framework 2.0, cannot be recorded. To check if your Framework 1.1 application is compatible, add the following XML tags to your application's .config file:

```
<configuration>
    <startup>
         <supportedRuntime version="v2.0.50727"/>
    </startup>
</configuration>
```

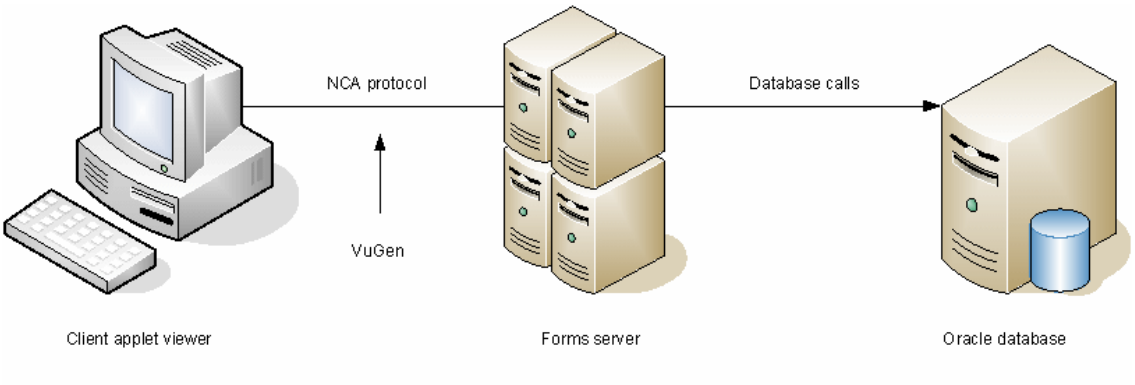
Invoke the application (without VuGen) and test its functionality. If the application works properly, VuGen can record it. Remove the above tags before recording the AUT with VuGen. For more information regarding this solution, see the MSDN Knowledge Base.

Oracle NCA Protocol

Oracle NCA Protocol Overview

Oracle NCA is a protocol that handles communication with the Oracle Forms server. Using your browser, you launch the database client, an applet viewer. You perform actions on the NCA database through its applet viewer. This eliminates the need for client software and allows you to perform database actions from all platforms that support the applet viewer.

The NCA environment is a three-tier environment. The user first sends an http call from his browser to a Web server. This call accesses the startup HTML page which invokes the Oracle Applications applet. The client (applet viewer) communicates through the proprietary NCA protocol with the application server (Oracle Forms server) which then submits information to the database server.



VuGen records and replays the NCA communication between the client and the Forms server (application server).

The Oracle NCA protocol is commonly used as a multi protocol in combination with Web (HTTP/HTML) or Web (Click & Script). This is the recommended way to record with Oracle NCA. If you are using Oracle NCA as a single protocol, web events are recorded but steps are not generated (or replayed) by default.

If you initially created a single protocol script for Oracle NCA, and at a later stage you require the Web functions for testing, you can regenerate your script in VuGen to add the Web functions, without having to re-record the session. You indicate this from the Protocols node in the Recording Options.

Oracle NCA Protocol Example Scripts

In the following example, the user selected an item from a list (**nca_list_activate_item**), pressed a button (**nca_button_press**), retrieved a list value (**nca_lov_retrieve_items**), and performed a click in an edit field (**nca_edit_click**). The logical names of the objects are the parameters of these functions.

```
...
nca_lov_select_item("Responsibilities","General Ledger, Vision
Operations");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","+ Journals");
nca_list_activate_item("FNDSCSGN.NAVIGATOR.LIST.0","       Enter");
nca_button_press("GLXJEENT.TOOLBAR.LIST.0");
nca_lov_find_value("Batches","");
```

```
nca_lov_retrieve_items("Batches",1,9);
nca_lov_select_item("Batches","AR 1020 Receivables 2537: A 1020");
nca_edit_click("GLXJEENT.FOLDER_QF.BATCH_NAME.0");
...
```

In certain tests, such as those performed on Oracle Configurator applications, information returned by one function is required throughout the session. VuGen automatically saves the dynamic information to a parameter, by inserting a **web_reg_save_param** function into the script. In the following example, the connection information is saved to a parameter called NCAJServSessionID.

```
web_reg_save_param ("NCAJServSessionId", "LB=\r\n\r\n", "RB=\r",
       LAST);
web_url("f60servlet",
         "URL=http://usscifforms05.sfb.na/servlet/f60servlet\?config
         =mult",          LAST);
```

In the above example, the right boundary is \r. The actual right boundary may differ between systems.

Note: We recommend that the user not modify the **web_reg_save_param** parameters if they were generated automatically. Alternatively, you can manually add a new **web_reg_save_param** function or add a new correlation rule.

Oracle NCA Record and Replay Tips

When recording an Oracle NCA Vuser script, follow these guidelines:

- We recommend installing Jinitiator before recording a script.
- Close all browsers before you begin recording.
- Record the login procedure in the **vuser_init** section. Record a typical business process in the Actions section. When you run the script, you can then specify multiple iterations for a specific business process. For more information, see "Solution Explorer - Overview" on page 50.
- VuGen supports the recording of Oracle Forms applications using the Forms Listener Servlet in multi-protocol mode. The application server uses the **Forms Listener Servlet** to create a runtime process for each client. The runtime process, **Forms Server Runtime**, maintains a persistent connection with the client and sends information to and from the server.

To support Forms 4.5 in replay, modify the **mdrv_oracle_nca.dat** file in the **dat > mdrv** folder to match the following example:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp110.dll
WIN95_EXT_LIBS=ncarp110.dll
LINUX_EXT_LIBS=liboranca.so
SOLARIS_EXT_LIBS=liboranca.so
HPUX_EXT_LIBS=liboranca.sl
AIX_EXT_LIBS=liboranca.so
```

```
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api
```

To restore Forms support for versions later than 4.5, restore the original values.

Pragma Mode

The client side of the Oracle NCA Vuser can be configured to send an additional header to the server named **Pragma**. The header is a counter that behaves in the following way: the initial message of the NCA handshake has a value of 1.

The messages that follow the handshake are counted, beginning with 3. The counter is incremented by 1 for each message sent by the client.

If the message received from the server is the type `plain\text` and the body of the message begins with `ifError:#/#00`, the client sends a 0 byte message to the server and the Pragma value changes its sign to a minus. This sign changes back after the client succeeds in receiving the information from the server.

Recording of the Pragma header is only supported in the multi-protocol mode (Oracle NCA and Web). You can identify the Pragma mode within the script's default.cfg file. When operating in Pragma mode, the UseServletMode is set to 2.

```
[HttpConnectMode]
UseHttpConnectMode=1
RelativeURL=<NCAJServSessionId>
UseServletMode=2
```

For information on the Pragma related run-time settings, see "Oracle NCA > Client Emulation Node" on page 379.

To identify the Pragma mode, you can perform a WinSock level recording and check the buffer contents. In the first example, the buffer contains the Pragma values as a counter:

```
send buf108
    "POST
/ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
    "vlet=gk5q79uqy1 HTTP/1.1\r\n"
    "Pragma: 1\r\n"
    ...
send buf110
    "POST
/ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
    "vlet=gk5q79uqy1 HTTP/1.1\r\n"
    "Pragma: 3\r\n"
    ...
```

In the following example, the buffer contains the Pragma values as an error indicator:

```
recv buf129 281
    "HTTP/1.1 200 OK\r\n"
    "Date: Tue, 21 May 2002 00:03:48 GMT\r\n"
    "Server: Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_
fastcgi/2.2"
```

```
    ".10 mod_perl/1.25 mod_oprocmgr/1.0\r\n"
    "Content-Length: 13\r\n"
    "Content-Type: text/plain\r\n"
    "\r\n"
    "ifError:8/100"
send buf130
    "POST
/ss2servlet/oracle.forms.servlet.ListenerServlet?JServSessionIdss2ser"
    "vlet=gk5q79uqy1 HTTP/1.1\r\n"
    "Pragma: -12\r\n"
    ...
```

How to Enable the Recording of Objects by Name

When recording an Oracle NCA script, you must record the session using object names instead of the standard object ID. If the script is recorded using the object ID, replay may fail because the ID is generated dynamically by the server and may differ between iterations. You can verify that your script is being recorded with object names by examining the **nca_connect_server** function.

```
nca_connect_server("199.35.107.119","9002"/*version=11i*/,
"module=/d1/oracle
        /visappl/fnd/11.5.0/forms/US/FNDSCSGN
userid=APPLSYSPUB/PUB@VIS
        fndnam=apps record=names ");
```

If the **record=names** argument does not appear in the **nca_connect_server** function, you may be recording object IDs. You can instruct VuGen to record object names by modifying one of the following:

Startup HTML File

If you have access to the startup HTML file, you instruct VuGen to record object names instead of its object ID by setting the **record=names** flag in the startup file, the file that is loaded when you start the Oracle NCA application. The following steps describe how to enable the recording of object names using the startup HTML file.

1. Edit the startup file that is called when the applet viewer begins by modifying the line shown below.

   ```
   <PARAM name="serverArgs ... fndnam=APPS">
   ```

2. Add the Oracle key **record=names** as shown below.

   ```
   <PARAM name="serverArgs ... fndnam=APPS record=names">
   ```

Forms Configuration File

If the application has a startup HTML file that references a Forms Web CGI configuration file **formsweb.cfg** (a common reference), you may encounter problems if you add **record=names** to the Startup file. In this situation, you should modify the configuration file. The following steps describe how to enable the recording of object names using the configuration file.

1. Go to the Forms Web CGI configuration file.

2. Define a new parameter in this file (see sample Web CGI configuration file below for this change).

```
serverApp=forecast
serverPort=9001
serverHost=easgdev1.dats.ml.com
connectMode=socket
archive=f60web.jar
archive_ie=f60all.cab
xrecord=names
```

3. Open the startup HTML file and locate `PARAM NAME="serverArgs"`.
4. Add the variable name as an argument to the ServerArgs parameter, for example, **record=%xrecord%** as shown below.

```
<PARAM NAME="serverArgs" VALUE="module=%form% userid=%userid%
%otherParams% record=%xrecord%">
```

5. Alternatively, you can edit the **basejini.htm** file in the Oracle Forms installation folder. This file is the default HTML file for running a form on the web using JInitiator-style tags to include the Forms applet. In the basejinin.htm file add the following line to the parameter definitions:

```
<PARAM NAME="recordFileName" VALUE="%recordFileName%">
```

In the <EMBED> tag, add the following line:

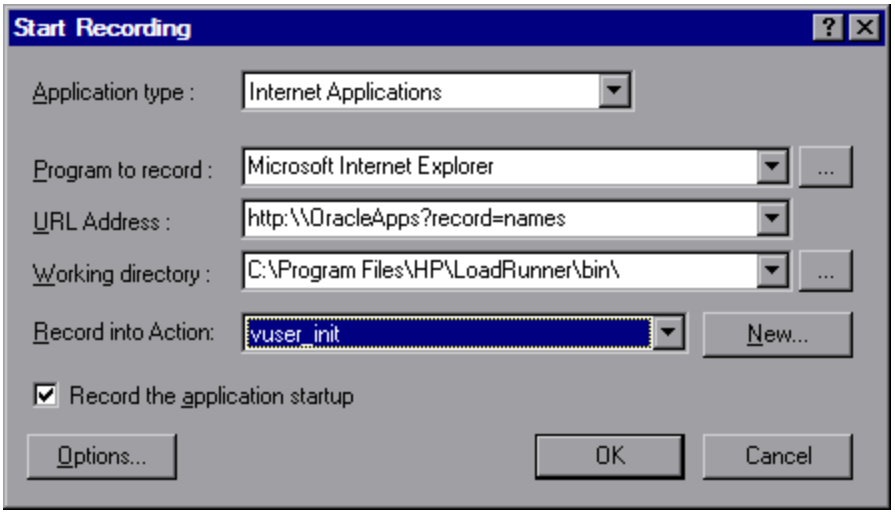
```
serverApp="%serverApp%"
logo="%logo%"
imageBase="%imageBase%"
formsMessageListener="%formsMessageListener%"
recordFileName="%recordFileName%"
```

The drawback in editing this file instead of the servlet configuration file **formsweb.cfg**, is that this file is replaced when you reinstall Oracle Forms. To avoid this, you can create a copy of the **basejini.htm** file and store it at another location. In the servlet configuration file, edit the **baseHTMLJinitiator** parameter to point to the new file.

URL to Record

If you do not have access to the startup HTML file, you can still have Oracle NCA record object names instead of its object ID by modifying the URL to record. The following solution only works if the startup HTML file does not reference another file while loading.

For this solution, you add "**?record=names**" after the URL in the Start Recording dialog box, after the URL name to record.



How to Launch Oracle Applications via the Personal Home Page

When launching Oracle Forms applications (versions 6i and higher) by logging in through the **Personal Home Page,** you must set several system profile options at the user level. It is desirable to pass such variables at the user level, and not at the site level, where it will affect all users. The following steps describe how to configure the "ICX: Forms Launcher" profile.

1. Sign on to the application and select the "System Administrator" responsibility.
2. Select **Profile/System** from the Navigator menu.
3. Within the **Find System Profile Values** form:
   a. Select the **Display > Site** option
   b. **Users** = <your user logon> for example, operations, mfg, and so on)
   c. **Enter Profile** =%ICX%Launch%
   d. Click **Find**.
4. Update the User value to the **ICX:Forms Launcher** profile:
   - If no parameter has been passed to the URL, append the following string to the end of the URL of the user value: **?**`play==;record=names`
   - If a parameter has been passed to the URL, append the following string to the end of the URL of the user value: **=;**`play==;record=names`
5. Save the transaction.
6. Log out of the Oracle Forms session.
7. Log out of the Personal Home Page session.
8. Sign on again via the **Personal Home Page** using your username.

If you were unable to update the ICX: Forms Launcher profile option at the user level, open the **Application Developer** responsibility and select the **Updatable** option for the ICX_FORMS_LAUNCHER profile.

The first parameter passed to the URL must begin with a question mark (?). You pass all subsequent parameters with an ampersand (=;). In most cases, the URL already contains parameters, which you can identify by searching for a question mark.

Oracle NCA - Troubleshooting and Limitations

This section describes troubleshooting and limitations for Oracle NCA protocol scripts.

Testing Secure Oracle NCA Applications

- In the Port Mapping node of the Recording Options dialog box, delete any existing entries for port 443 and create a new entry for the Oracle server name:

Service ID: HTTP
Target Server: Oracle Forms Server IP address or long host name
Target Port: 443
Connection Type: SSL
SSL Version: Active SSL version. If in doubt, select SSL 2/3.

Note that the **Service ID** is **HTTP** and not **NCA**.

For more information, see "Network > Port Mapping Node" on page 313.

- If you encounter problems when replaying an NCA HTTPS script during the **nca_connect_server** command, insert the following function at the beginning of the script.

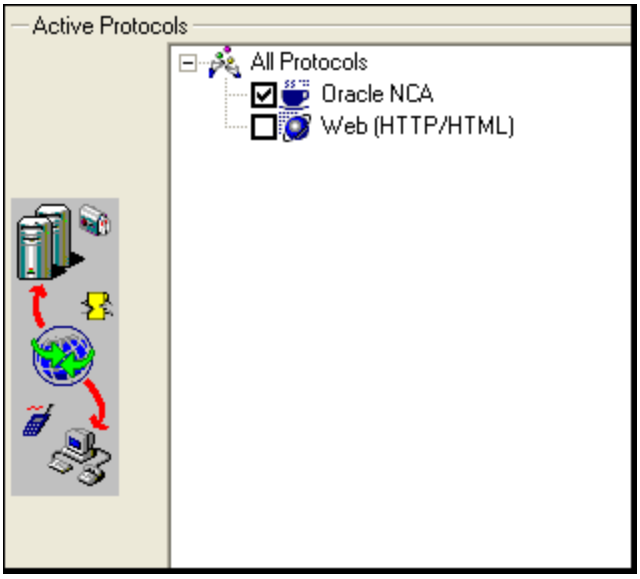
```
web_set_sockets_option("SSL_VERSION","3");
```

Testing Servlets and other Oracle NCA Applications

Certain NCA sessions use servlets:

- the Forms Listener servlet
- applications or modules that use both NCA and HTTP communications, such as the Oracle Configurator
- the initializing of the NCA application (downloading the applet, jar, and gif files)

When recording servlets, you must record both Oracle NCA and Web functions. You can do this by using the Oracle Apps IIi protocol or creating an Oracle NCA multi-protocol script. Alternatively, if you created a single protocol script for Oracle NCA, open the **General > Protocols** node in the Recording Options, and enable the Web protocol. Then you can begin recording.



If you are unsure whether your application uses servlets, you can check the **default.cfg** file in the script folder after recording a script. Locate the entry "**UseServletMode="**

If the value is 1 or 2, then servlets are being used and you must enable HTTP recording in addition to Oracle NCA.

If you already recorded a script, you can regenerate the code automatically to include the Web functions without having to re-record. Select **Record > Regenerate Script**, and select the Web protocol in the Protocols section.

Determining the Recording Mode

When recording Oracle NCA scripts: VuGen automatically determines the correct connection mode: HTTP or Socket mode. Generally, you are not required to modify any of the recording settings as VuGen auto-detects the system configuration (unless you are working with Forms Server 4.5). In systems where the standard port mapping are reserved by other applications, you may need to modify the Port Mapping settings, depending on the recording mode.

You can determine the recording mode in one of the following ways:

- When using the NCA application, open the Java Console.

```
proxyHost=null
proxyPort=0
connectMode=HTTP
Forms Applet version is: 60812
```

The **connectMode** entry indicates **HTTP**, **HTTPS**, or **socket**.

- After recording an NCA session, open the **default.cfg** file in the Vuser folder and check the value of the **UseHttpConnectMode** entry.

```
[HttpConnectMode]
UseHttpConnectMode= 2
// 0 = socket 1 = http 2 = https
```

When defining a new port mapping in the Server Entry dialog box, use a **Service ID** of HTTP for HTTP or HTTPS modes. For Socket mode, use a **Service ID** of NCA.

For more information about Port Mapping settings, see "Network > Port Mapping Node" on page 313.

Recording Trace Information for Oracle DB

To debug your script, you can use the Oracle DB breakdown graphs. To gather data for this graph, you turn on the trace mechanism before running the script.

To manually turn on the tracing mechanism, use the **nca_set_custom_dbtrace** function. For more information, see the Function Reference (**Help > Function Reference**).

RDP Protocol

RDP Protocol - Overview

The Microsoft RDP (Remote Desktop Protocol) allows users to connect to computers remotely . For example, users can use RDP to connect to a central and powerful server for working on specific business applications or graphic terminals. This provides users with the same look and feel as if they are working on standalone PCs.

Note: RDP versions 5.1 and later have an **Experience** tab that allows you to set various options. This tab is not supported by VuGen recording. All options are set to the ON position.

RDP Recording Tips

When recording an RDP Vuser script, follow these guidelines in order to create an effective script.

Single vs. Multi-Protocol Scripts

When creating a new script, you may create a single protocol or multi-protocol script. For example, to record both RDP traffic and Web responses, create a multi-protocol script for RDP and Web to enable the recording of both protocols.

Record into Appropriate Sections

Record the connection process into the **vuser_init** section, and the closing process into the **vuser_end** section. This will prevent you from performing iterations on the connecting and disconnecting processes. For more information about recording into sections, see "Solution Explorer - Overview" on page 50.

Run a Clean Session

When recording a session, make sure to perform the complete business process, starting with the connection and ending with the cleanup. End your session at a point from where you could start the entire process from the beginning. Do not leave any client or application windows open.

You should also configure your terminal server to end disconnected sessions. Select **Administrative Tools > Terminal Services Configuration > Connection Properties > Sessions > Override User Settings** and set the server to end disconnected sessions.

Explicit Clicks

When opening expanded menu options, click explicitly on each option—do not depend on the expanding menu. For example, when choosing **Start > All Programs > Microsoft Word**, be sure to click on the word **Programs**.

Working with Clipboard Data (RDP Protocol)

VuGen allows you to copy and paste the text of a clipboard during an RDP session. You can copy the contents locally and paste them remotely, or vice versa—copy the contents from the remote machine and paste them locally. The copying of text is supported in TEXT, LOCALE, and UNICODE formats.

VuGen generates separate functions when copying or saving the clipboard data.

Code sample #1

The following example illustrates a copy operation on a local machine and a paste operation on a remote machine:

```
//Notifies the Remote Desktop that new data is available in the Local
machine's
//clipboard.The data can be provided in three formats: TEXT, UNICODE
and LOCALE
rdp_notify_new_clipboard_data(
"StepDescription=Send local clipboard formats 1",
"Snapshot=snapshot1.inf",
"FormatsList=RDP_CF_TEXT|RDP_CF_UNICODE|RDP_CF_LOCALE",
RDP_LAST );
rdp_key(
"StepDescription=Key Press 2",
"Snapshot=snapshot_9.inf",
"KeyValue=V",
"KeyModifier=CONTROL_KEY",
RDP_LAST );
//Provides clipboard data to the Remote Desktop when it requests the
data.
rdp_send_clipboard_data(
"StepDescription=Set Remote Desktop clipboard 1",
"Snapshot=snapshot1.inf",
"Timeout=20",
REQUEST_RESPONSE, "Format=RDP_CF_UNICODE", "Text=text for clipboard",
RDP_LAST);
```

Code sample #2

This example illustrates a copy operation on a remote machine and a paste operation on a local machine:

```
rdp_key(
"StepDescription=Key Press 2",
"Snapshot=snapshot_9.inf",
"KeyValue=C",
"KeyModifier=CONTROL_KEY",
```

```
RDP_LAST);
// The function requests the Remote Desktop UNICODE text and saves it
to a //parameter
rdp_receive_clipboard_data(
"StepDescription=Get Remote Desktop clipboard 1",
"Snapshot=snapshot1.inf",
"ClipboardDataFormat=RDP_CF_UNICODE",
"ParamToSaveData=MyParam",
RDP_LAST);
```

Normally, the Remote Desktop clipboard data is saved in UNICODE format. If the Remote Desktop requests data in the TEXT or LOCALE formats, the **rdp_send_clipboard_data** function automatically converts the content of MyParam from UNICODE into the requested format and sends it to the Remote Desktop. The Replay log indicates this conversion with an informational message. If the conversion is not possible, the step fails.

For more information about the rdp functions, see the Function Reference (**Help > Function Reference**).

Correlating Clipboard Parameters

During a recording session, if the client sends the server the same data that it received, VuGen replaces the sent data with a parameter during code generation. VuGen performs this correlation only when the received and sent data formats are the same.

Code sample #3

The following example shows how the same parameter, **MyParam**, is used for both receiving and sending the data.

```
// Receive the data from the server
rdp_receive_clipboard_data("StepDescription=Get Remote Desktop
clipboard 1",
"Snapshot=snapshot_9.inf",
"Timeout=0",
"ClipboardDataFormat=RDP_CF_UNICODETEXT",
"ParamToSaveData=MyParam",
RDP_LAST);
...
// Send the data to the server
rdp_send_clipboard_data("StepDescription=Get Remote Desktop clipboard
1",
"Snapshot=snapshot_9.inf",
"Timeout=10",
REQUEST_RESPONSE, "Format=RDP_CF_UNICODETEXT", "Text={MyParam}",
RDP_LAST);
```

RDP Snapshots - Overview

Vuser scripts based on the RDP protocol utilize VuGen's Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

When you open an RDP Vuser script, VuGen's standard Snapshot pane functionality is available. The Snapshot pane displays snapshots of the remote display, saved during recording and playback of the Vuser script. Typically, these snapshots are used to synchronize playback of the Vuser script.

In addition to the basic Snapshot pane functionality, the Snapshot pane for RDP Vuser scripts lets you display snapshots in one of the following views:

- **Image.** Displays only the image of the snapshot and is ideal for visually comparing two images. This view displays the snapshot faster and requires less memory than the Full view. You can synchronize two snapshots in the Snapshot pane if both snapshots are displayed in the Image view. The Image view does not automatically scroll to show the area of a snapshot that is used for synchronization.
- **Full.** Scrolls to display the area that is used for synchronization. This view displays the snapshot slower and requires more memory than the Image view. You cannot synchronize two snapshots displayed in the Snapshot pane if either of the snapshots is displayed in the Full view. By default, snapshots are displayed in the Full view.

To display a specific synchronization snapshot in the Snapshot pane, do one of the following:

- In the Editor, select the step that contains a reference to the snapshot.
- In the Step Navigator, double-click the step that contains a reference to the snapshot.

When working with RDP Vuser scripts, the Snapshot pane lets you copy a snapshot to the clipboard, and display a snapshot of the most recent replay error. For more information on how to use the Snapshot pane, see "How to Work with Snapshots" on page 57.

Image Synchronization Overview

An RDP session executes remotely. All keyboard and mouse handling is done on the server, and it is the server that reacts to them. For example, when you double-click an application on the desktop, it is the server that realizes a double-click took place and that the application must be loaded and displayed.

When an RDP client connects to a server, it does two things:

- It sends the server coordinates of actions. For example, 'clicked the left mouse button at coordinates (100, 100) on the screen'.
- It receives images from the server showing the current status of the screen after the action took place

The RDP client (and therefore, LoadRunner) does not know that the screen contains windows, buttons, icons, or other objects. It only knows the screen contains an image and at what coordinates the user performed the action. To allow the server to correctly interpret the actions, you set synchronization points within the script. These points instruct the script to wait until the screen on the server matches the stored screen before continuing.

The next time you replay the script, it will wait until the image returned by the server matches the image you selected.

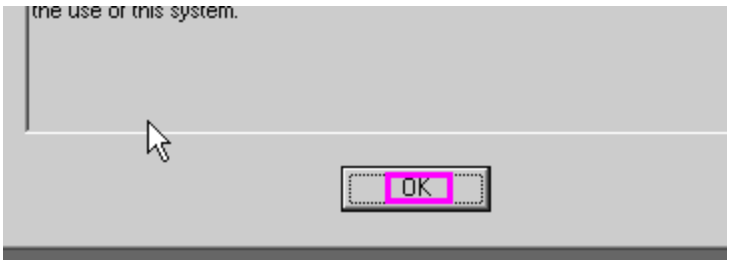
Image Synchronization Tips (RDP Protocol)

Use the following guidelines for effective image synchronization:

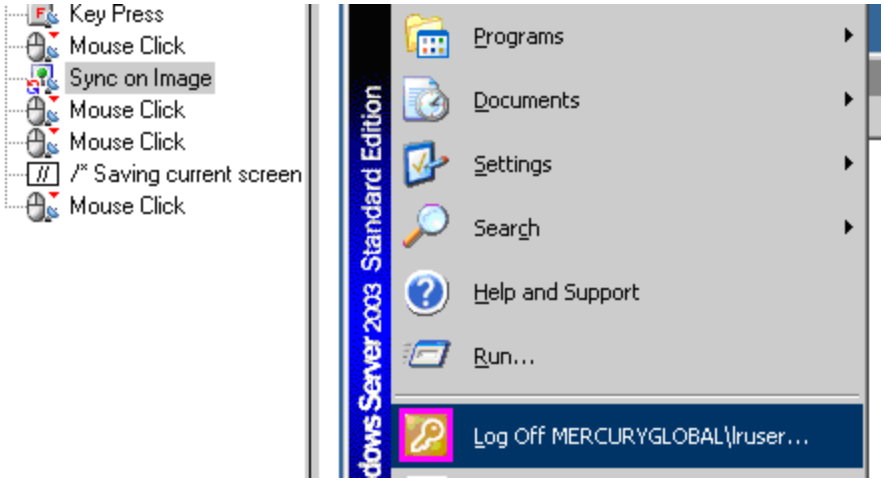
Synchronize on the Smallest Significant Area

When synchronizing on an image, try to synchronize only the part of the image that is necessary. Additional details within the image may not be reproduced during replay and could result in a synchronization failure.

For example, when synchronizing on an image of a button, select only the text itself and not the dotted lines around the text as they may not appear during replay.



When synchronizing a highlighted area, try to capture only the part of the image that is not effected by the highlighting. In the following example, perform a synchronization on the Log Off icon, but not the entire button, since the highlighting may not appear during replay, and the color could vary with different color schemes.



Synchronize Before Every User Action

It is recommended that you synchronize before every mouse operation. You should also synchronize before the first **rdp_key** or **rdp_type** operation that follows a mouse operation.

Image Synchronization - Shifted Coordinates (RDP Protocol)

When replaying a script, a recorded object may appear at different coordinates on the screen. The object is the same, but its placement has been shifted. For example, during recording a window opened at coordinates (100, 100), but during replay at (200, 250).

In this case, the synchronization point will automatically find the new coordinates without any intervention on your part. It will automatically note the difference of 100 pixels in the horizontal axis and 150 pixels in the vertical axis.

All subsequent mouse operations that are coordinate dependent will use the modified coordinates, so that a mouse click recorded at (130, 130) will be replayed to (230, 280) = (130 + 100, 130 + 150).

You control the shifting of the coordinates through the **AddOffsetToInput** parameter in the **rdp_sync_on_image** step. You can override this parameter to either add or not add the differences in location during replay to the recorded coordinates for any further operations. If you do not override this parameter, VuGen takes its value from the default setting in the run-time settings.

The corresponding parameter in the operations (for example **rdp_mouse_click** or **rdp_mouse_drag**) is **Origin**. This parameter decides whether the operation should take its coordinates only from the 'clean' values that were recorded, or whether it should take into account the differences that were added by the last synchronization point. If not explicitly specified, VuGen takes the value for this parameter from the run-time settings.

RDP Agent (Agent for Microsoft Terminal Server) Overview

The Agent for Microsoft Terminal Server is an optional utility that you can install on the RDP server. It provides enhancements to the normal RDP functionality. It is provided in the LoadRunner installation DVD and you can install it on any RDP server. The agent provides you with more intuitive and readable scripts, built-in synchronization, and detailed information about relevant objects. Note that when you run RDP Vusers with the agent installed, each Vuser runs its own process of lrrdpagent.exe. This results in a slight reduction in the number of Vusers that can run on the server machine.

Tips for Using the Agent for Microsoft Terminal Server

- When opening application menus (e.g. File, Edit...) with the mouse, sync steps will sometimes fail. To avoid this issue, use the keyboard to select menu items when recording.
- When you add a **sync_on_object_mouse_click** step manually, the coordinates given are absolute coordinates (relating to the entire screen). To create the synchronization point, you need to calculate the offset in the window (relative coordinates) of the desired click location and modify the absolute coordinates accordingly for the synchronization to successfully replay.
- If a synchronization object exists at the correct location and time during replay, but is covered by another window (such as a pop-up), then the synchronization step will pass and a click will be executed on the window which is covering the synchronization point and therefore harm the script flow.

- During recording, if you want to return the application window to the foreground, either click on the title bar, or use the keyboard (ALT+TAB). Note that if you click inside the application window to return it to the foreground, the RDP session may terminate unexpectedly.

The Agent for Microsoft Terminal Server provides the following enhancements to the normal RDP functionality:

Object Detail Recording

When the Agent for Microsoft Terminal Server is installed, VuGen can record specific information about the object that is being used instead of general information about the action. For example, VuGen generates **sync_object mouse_click** and **sync_object_mouse_double_click** steps instead of **mouse_click** and **mouse_double_click** that it generates without the agent.

The following example shows a double-mouse-click action recorded with and without the agent installation. Note that with the agent, VuGen generates sync_object functions for all of the mouse actions.

```
rdp_sync_object_mouse_double_click("StepDescription=Mouse Double Click
on Synchronized Object 1",
        "Snapshot=snapshot_12.inf",
        "WindowTitle=RDP2",
        "Attribute=TEXT",
        "Value=button1",
        "MouseX=100",
        "MouseY=71",
        "MouseButton=LEFT_BUTTON",
        RDP_LAST);
rdp_mouse_double_click("StepDescription=Mouse Double Click 1",
        "Snapshot=snapshot_2.inf",
        "MouseX=268",
        "MouseY=592",
        "MouseButton=LEFT_BUTTON",
        "Origin=Default",
        RDP_LAST);
```

Expanded Right-Click Menu

When you click within a snapshot, you can insert several functions into the script using the right-click menu. When the agent is not active, you are limited to inserting only **mouse_click**, **mouse_double_click**, and **sync_on_Image** steps. When the agent is installed, you are able to insert all possible steps that involve the RDP agent:

- **get_object_info** and **Sync_on_object_iInfo**. Provides information about the state of the object, and synchronize on a specific object property such as: ENABLED, FOCUSED, CONTROL_ID, ITEM_TEXT, TEXT, CHECKED, and LINES.
- **insert_sync_on_text** and **get_text**. For details, see the Function Reference (**Help > Function Reference**).

Code sample

In the following example, the **rdp_sync_on_object_info** function provides synchronization by waiting for the Internet Options dialog box to come into focus.

```
rdp_sync_on_object_info("StepDescription=Sync on Object Info 0",
        "Snapshot=snapshot_30.inf",
        "WindowTitle=Internet Options",
        "ObjectX=172",
        "ObjectY=155",
        "Attribute=FOCUSED",
        "Value={valueParam}",
        "Timeout=10",
        "FailStepIfNotFound=No",
        RDP_LAST);
```

How to Install / Uninstall the RDP Agent

The installation file for the Agent for Microsoft Terminal Server is located on the LoadRunner installation disk, under the **Additional Components\ Agent for Microsoft Terminal Server** folder.

> Note that the agent should be installed on your RDP server machine only, not on Load Generator machines.

If you are upgrading the agent, make sure to uninstall the previous version before installing the new one (see uninstallation instructions below).

Install the RDP Agent

1. If your server requires administrator permissions to install software, log in as an administrator to the server.
2. If you are using a Remote Desktop connection (RDP) to install the agent onto a machine running Windows 2003, run the following command on the target machine before starting the installation:

   ```
   Change user /install
   ```

3. Locate the installation file, **Setup.exe**, on the LoadRunner DVD in the **Additional Components\ Agent for Microsoft Terminal Server** folder.
4. Follow the installation wizard to completion.

To use the agent, you must set the recording options before recording a Vuser script. In the Start Recording dialog box, click **Options**. In the Advanced Code Generation node, select the **Use RDP Agent** check box.

Uninstall the RDP Agent

1. If your server requires administrator privileges to remove software, log in to the server as an administrator.
2. Select **Control Panel > Add/Remove Programs > HP Software Agent for Microsoft Terminal Server** and click **Change/Remove**.

How to Add Image Synchronization Points to a Script

1. Select an operation to which you would like to add a synchronization point in your script.
2. Right-click on the image snapshot and select Insert Synch On Image from the menu. The cursor will change to a cross-hair.
3. Mark the area on the screen that you would like to synchronize upon by clicking on the left button and dragging the box to enclose the area. When you release the mouse button, the Sync on Image dialog box opens.
4. Click **OK**. VuGen adds a new Sync on Image step before the selected step. When you select this step, VuGen displays a snapshot that contains a pink box around the area you selected for synchronization.

The next time you replay the script, it will wait until the image returned by the server matches the image you selected.

Failed Image Synchronization Dialog Box (RDP Protocol)

This dialog box opens when an image synchronization fails during the replay of a script. You can stop the script or continue the replay despite the error.

To access	Opens automatically when an image synchronization fails.
Important information	**The content of this dialog box varies depending on the reason for the failed synchronization:** • **Append Snapshot.** The Failed Image Synchronization - Append Snapshot dialog box opens when the replay image is so different from the record image that changing the tolerance level will not help. • **Raise Tolerance.** The Failed Image Synchronization - Raise Tolerance dialog box opens when the script replay failed to find the exact image requested, but if the tolerance level for performing synchronization on images was relaxed, then it would have succeeded in finding the image. • **Lower Tolerance.** The Failed Image Synchronization - Lower Tolerance dialog box opens when the script replay fails to meet the NotAppear or Change conditions. VuGen detected an image match where you expected it not to detect one. If the tolerance level was reduced, the recorded and replay images would not match, and the NotAppear or Change conditions would be met resulting in a successful replay. • **Non Specified.** The Failed Image Synchronization dialog box opens when the script replay fails to meet any of the synchronization conditions such as NotAppear or Change. VuGen did not find another image at the original coordinates that could be appended to the script.
See also	Image Synchronization Overview

User interface elements are described below:

UI Element	Description
Stop	Consider the mismatch between the snapshots to be an error. This error will be handled like all other errors and halt the execution of the script.
Continue	This button performs different actions depending on the type of dialog box: • **Append Snapshot.** Accept the mismatch. VuGen appends the replay snapshot as a "record" snapshot for the step. In future replays of the step, VuGen uses all existing record snapshots and the appended snapshot as the basis for comparison between screens. If the replay returns any of the record snapshots, the Vuser will not fail. You can view the original and appended snapshots for a step by clicking the navigation arrows  in the Snapshot pane toolbar. • **Lower Tolerance.** Accept the mismatch and lower the tolerance level so that VuGen permits a smaller mismatch between the record images and those displayed during replay. • **Raise Tolerance.** Accept the mismatch and raise the tolerance level so that VuGen permits a greater mismatch between the record images and those displayed during replay. • **Non-specified.** Accept the mismatch, and do not make any changes in the script. Continue script execution despite the mismatch. **Note:** Raising or lowering the tolerance level from the dialog box changes the level for the current step only. To change the tolerance level for the whole script, change the **Default tolerance for image synchronization** setting in the **Run-Time Settings > RDP > Synchronization** node.

RDP - Troubleshooting and Limitations

This section describes troubleshooting information for RDP scripts using the Agent for Microsoft Terminal Server.

Replay fails on rdp_sync_object_mouse_click/double_click steps.

If the replay fails on specific **rdp_sync_object_mouse_click** or **rdp_sync_object_mouse_double_click** steps, there are workarounds to resolve the issue. We recommend that you try the workarounds in the order they are listed.

Workaround: Modify RDPAgentCodeGen.cfg file

The **RDPAgentCodeGen.cfg** file can configure VuGen to automatically create an **rdp_sync_on_image** and **rdp_mouse_click** step the next time the script is generated for each **rdp_sync_object_mouse_click/double_click** steps which occur within a given window. To do this, you specify the name of the window, update a variable which counts the total number of windows for which this process occurs, and regenerate the script.

Modify the RDPAgentCodeGen.cfg file

1. Open the **RDPAgentCodeGen.cfg file** in the **Script Directory > data** folder.
2. Open the **Step Navigator** and double-click the step that failed.

3. Copy the name of the window
4. In the **RDPAgentCodeGen.cfg** file, increase the value of **NumberOfTitles** by 1.
5. Add a line as follows:

   ```
   WindowTitleX=<name of window>
   ```

   where **X** is the new value of **NumberOfTitles**.
6. Regenerate the script.

> **Note:** The **RDPAgentCodeGen.cfg file** can be used to automatically produce **rdp_sync_on_image** and **rdp_mouse_click** steps in a similar way for **rdp_sync_object_mouse_click/double_click** steps which are specified in different ways as well. Steps can be targeted based on the class attribute of the control. For more information, contact HP software support.

Workaround: Manually Insert a New Step

You can manually perform the workaround, by inserting an **rdp_sync_on_image** and **rdp_mouse_click** step for each step that fails. We do not recommend using this workaround because steps added in this way will be lost if the script is regenerated.

RTE Protocol

RTE Protocol Overview

An RTE Vuser types character input into a terminal emulator, submits the data to a server, and then waits for the server to respond. For instance, suppose that you have a server that maintains customer information for a maintenance company. Each time a field service representative makes a repair, he accesses the server database by modem using a terminal emulator. The service representative accesses information about the customer and then records the details of the repair that he performs.

You could use RTE Vusers to emulate this case. An RTE Vuser would:

1. Type **60** at the command line to open an application program.
2. Type **F296**, the field service representative's number.
3. Type **NY270**, the customer number.
4. Wait for the word "Details" to appear on the screen. The appearance of "Details" indicates that all the customer details are displayed on the screen.
5. Type **Changed gasket P249, and performed Major Service** the details of the current repair.
6. Type **Q** to close the application program.

You use VuGen to create RTE Vuser scripts. The script generator records the actions of a human user on a terminal emulator. The script generator records the keyboard input from the terminal window, generates the appropriate statements, and inserts them into the Vuser script. While you

record, the script generator automatically inserts synchronization functions into the script. For details, see "RTE Synchronization Overview" on page 622.

The functions developed to emulate a terminal communicating with a server are called TE Vuser functions. Each TE Vuser function has a **TE** prefix. VuGen automatically records most of the TE functions listed in this section during an RTE recording session. You can also manually program any of the functions into your script**.**

For syntax and examples of the TE functions, see the Function Reference (**Help > Function Reference**).

An RTE Vuser emulates the actions of a real user. Human users use terminals or terminal emulators to operate application programs.

In the RTE Vuser environment, a Vuser replaces the human. The Vuser operates PowerTerm, a terminal emulator.

PowerTerm works like a standard terminal emulator, supporting common protocols such as IBM 3270 =; 5250, VT100, VT220, and VT420-7.

Working with Ericom Terminal Emulation

VuGen supports record and replay with Ericom Terminal Emulators.

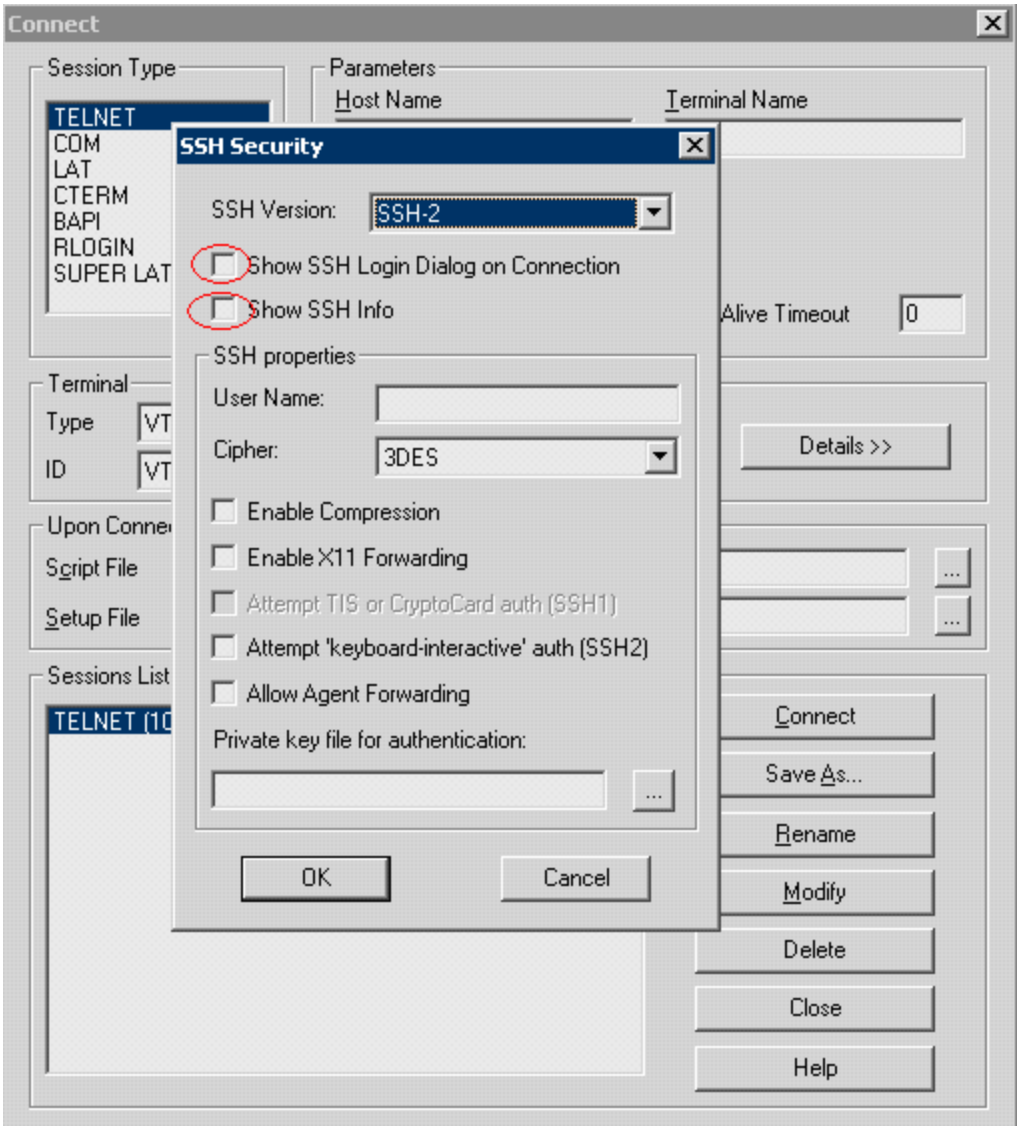
The Ericom support handles escape sequences during record and replay. Ericom's PowerTerm lets you map PC keys to custom escape sequences. For information about mapping, see the PowerTerm help.

When a user presses mapped keys while recording an Ericom VT session, VuGen generates **TE_send_text** functions instead of the standard **TE_type**. This allows the script to handle custom escape sequences in a single step. For more information, see the Function Reference (**Help > Function Reference**) for the **TE_send_text** function.

SSL and SSH Support for Ericom

VuGen also supports SSL/SSH record and replay for the RTE Ericom library. To work with SSL or SSH, you select the type in the **Security** section of the Connect dialog box.

When working with SSH Security, by default VuGen opens a popup dialog box prompting you for more information. We recommend that you disable the **Show** options to prevent the pop-ups from being issued. If you enable these pop-ups, it may affect the replay. You can access the advanced security options by clicking the **Details** button.



Typing Input into a Terminal Emulator

Two TE Vuser functions enable Vusers to "type" character input into the PowerTerm terminal emulator:

- **TE_type** sends characters to the terminal emulator. When recording, the VuGen automatically generates **TE_type** functions for keyboard input to the terminal window. For details, see below.
- **TE_typing_style** determines the speed at which the Vuser types. You can manually define the typing style by inserting a **TE_typing_style** function into the Vuser script. For details, see "Typing Input into a Terminal Emulator" above. Alternatively, you can set the typing style by using the run-time settings. For more information, see "Run-Time Settings" on page 332.

> **Note:** While recording an RTE Vuser script, do not use the mouse to relocate the cursor within the terminal emulator window. VuGen does not record these cursor movements.

Using the TE_type Function

When you record a script, the VuGen records all keyboard input and generates appropriate **TE_type** functions. During execution, **TE_type** functions send formatted strings to the terminal emulator.

Keyboard input is defined as a regular text string (including blank spaces). For example:

```
TE_type ("hello, world");
```

Input key names longer than one character are represented by identifiers beginning with the letter k, and are bracketed within greater-than/less-than signs (< >).

For example, the following function depicts the input of the Return key followed by the Control and y keys:

```
TE_type("<kReturn><kControl-y>");
```

Some other examples include: <kF1>, <kUp>, <kF10>, <kHelp>, <kTab>.

To determine a key name, record an operation on the key, and then check the recorded statement for its name.

Note: When you program a **TE_type** statement (rather than recording it), use the key definitions provided in the Function Reference (**Help > Function Reference**).

Setting the Timeout Value for TE_type

If a Vuser attempts to submit a **TE_type** statement while the system is in X SYSTEM (or input inhibited) mode, the Vuser will wait until the X SYSTEM mode ends before typing. If the system stays in X SYSTEM mode for more than TE_XSYSTEM_TIMEOUT milliseconds, then the **TE_type** function returns a TE_TIMEOUT error.

You can set the value of TE_XSYSTEM_TIMEOUT by using **TE_setvar**. The default value for TE_XSYSTEM_TIMEOUT is 30 seconds.

Allowing a Vuser to Type Ahead

Under certain circumstances you may want a Vuser to submit a keystroke even though the system is in X SYSTEM (or input inhibited) mode. For example, you may want the Vuser to press the `Break` key. You use the TE_ALLOW_TYPEAHEAD variable to enable the Vuser to submit a keystroke even though the system is in X SYSTEM mode.

Set TE_ALLOW_TYPEAHEAD to zero to disable typing ahead, and to any non-zero number to permit typing ahead. You use **TE_setvar** to set the value of TE_ALLOW_TYPEAHEAD. By default, TE_ALLOW_TYPEAHEAD is set to zero, preventing keystrokes from being sent during X SYSTEM mode.

For more information about the **TE_type** function and its conventions, see the Function Reference (**Help > Function Reference**).

Setting the Typing Style

You can set two typing styles for RTE Vuser: FAST and HUMAN. In the FAST style, the Vuser types input into the terminal emulator as quickly as possible. In the HUMAN style, the Vuser

pauses after typing each character. In this way, the Vuser more closely emulates a human user typing at the keyboard.

You set the typing style using the **TE_typing_style** function. The syntax of the **TE_typing_style** function is:

int TE_typing_style (char * style);

where style can be FAST or HUMAN. The default typing style is HUMAN. If you select the HUMAN typing style, the format is:

HUMAN, delay [, first_delay]

The delay indicates the interval (in milliseconds) between keystrokes. The optional parameter first_ delay indicates the wait (in milliseconds) before typing the first character in the string. For example,

```
TE_typing_style ("HUMAN, 100, 500");
TE_type ("ABC");
```

means that the Vuser will wait 0.5 seconds before typing the letter A; it will then wait 0.1 seconds before typing "B" and then a further 0.1 seconds before typing "C".

For more information about the **TE_typing_style** function and its conventions, see the Function Reference (**Help > Function Reference**).

In addition to setting the typing style by using the **TE_typing_style** function, you can also use the run-time settings. For details, see "Run-Time Settings" on page 332.

Generating Unique Device Names

Some protocols, such as APPC, require a unique device name for each terminal that logs on to the system. Using the run-time settings, you can specify that the TE_connect function generate a unique 8-character device name for each Vuser, and connect using this name. Although this solves the requirement for uniqueness, some systems have an additional requirement: The device names must conform to a specific format. See "Run-Time Settings" on page 332 for more information.

To define the format of the device names that the TE_connect function uses to connect a Vuser to the system, add an **RteGenerateDeviceName** function to the Vuser script. The function has the following prototype:

```
void RteGenerateDeviceName(char buf[32])
```

The device name should be written into **buf**.

If an **RteGenerateDeviceName** function exists in a Vuser script, the Vuser calls the function each time a new device name is needed. If no **RteGenerateDeviceName** function is defined in the script—and unique device names are required—the **TE_connect** function generates the required names.

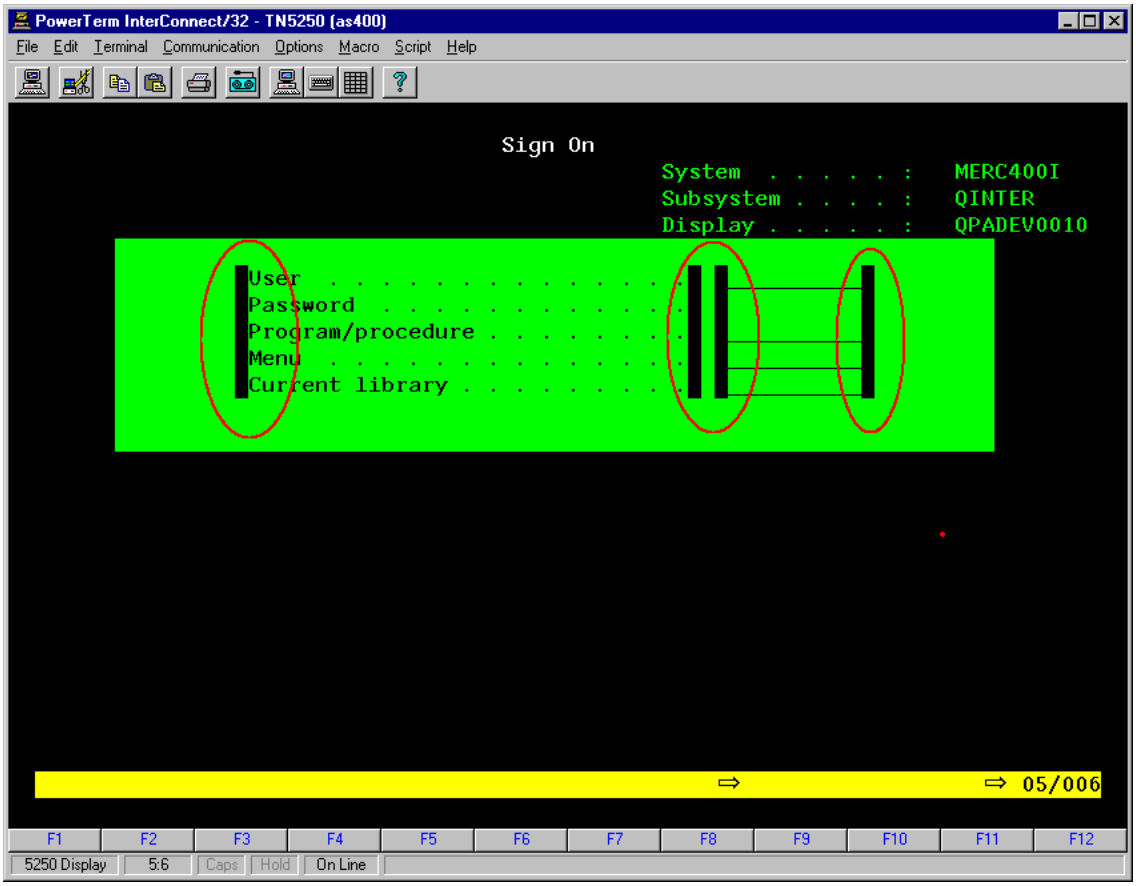
In the following example, the RteGenerateDeviceName function generates unique device names with the format "TERMx". The first name is TERM0, followed by TERM1, TERM2, and so forth.

```
RteGenerateDeviceName(char buf[32])
{
    static int n=0;
    sprintf(buf, "TERM%d", n);
```

```
    n=n+1;
}
```

Setting the Field Demarcation Characters

Some terminal emulators use demarcation characters to mark the beginning and the end of each field. These demarcation characters are not visible—appearing on the screen as spaces. In the terminal emulator shown below, the colors in the middle section of the screen have been inverted to display the field demarcation characters. These characters are surrounded by ellipses.



The **TE_wait_text**, **TE_get_text**, and **TE_find_text** functions operate by identifying the characters in a specified portion of the screen. If a field demarcation character is located within the specified section, you can identify the character as a space or an ASCII character. You use the TE_FIELD_CHARS system variable to specify the method of identification. You can set TE_FIELD_CHARS to 0 or 1:

- 0 specifies that the character in the position of the field demarcation characters is returned as a space.
- 1 specifies that the character in the position of the field demarcation characters is returned as an ascii code (ascii 0 or ascii 1).

By default, TE_FIELD_CHARS is set to 0.

You retrieve and set the value of TE_FIELD_CHARS by using the **TE_getvar** and **TE_setvar** functions.

Reading Text from the Terminal Screen

There are several Vuser functions that RTE Vusers can use to read text from the terminal screen. You can use these functions, **TE_find_text** and **TE_get_text_line**, to check that the terminal emulator is responding correctly, or to enhance the logic in your scripts.

After recording, you can manually insert **TE_find_text** and **TE_get_text_line** statements directly into your RTE Vuser scripts.

Searching for Text on the Screen

The **TE_find_text** function searches for a line of text on the screen. The syntax of the function is:

```
int TE_find_text (char *pattern, int col1, int row1, int col2, int row2,
        int *retcol, int *retrow, char *match );
```

This function searches for text matching pattern within the rectangle defined by col1, row1, col2, row2. Text matching the pattern is returned to match, and the actual row and column position is returned to retcol and retrow. The search begins in the top-left corner. If more than one string matches pattern, the one closest to the top-left corner is returned.

The **pattern** can include a regular expression. See the Function Reference (**Help > Function Reference**) for details on using regular expressions.

You must manually type **TE_find_text** statements into your Vuser scripts. For details on the syntax of the **TE_find_text** function, see the Function Reference (**Help > Function Reference**).

Reading Text from the Screen

The **TE_get_text_line** function reads a line of text from the area of the screen that you designate. The syntax of the function is:

```
char *TE_get_text_line (int col, int row, int width, char * text );
```

This function copies a line of text from the terminal screen to a buffer text. The first character in the line is defined by col, row. The column coordinate of the last character in the line is indicated by width. The text from the screen is returned to the buffer text. If the line contains tabs or spaces, the equivalent number of spaces is returned.

In addition, the **TE_get_cursor_position** function can be used to retrieve the current position of the cursor on the terminal screen. The **TE_get_line_attribute** function returns the character formatting (for instance, bold or underline) of a line of text.

You must manually type **TE_get_text_line** statements into your Vuser scripts. For details on the syntax of the **TE_get_text_line** function, see the Function Reference (**Help > Function Reference**).

RTE Synchronization Overview

Depending on the system you are testing, you may need to synchronize the input that a Vuser sends to a terminal emulator with the subsequent responses from the server. When you synchronize input, you instruct the Vuser to suspend script execution and wait for a cue from the

system, before the Vuser performs its next action. For instance, suppose that a human user wants to submit the following sequence of key strokes to a bank application:

1. Type 1 to select "Financial Information" from the menu of a bank application.
2. When the message "What information do you require?" appears, type 3 to select "Dow Jones Industrial Average" from the menu.
3. When the full report has been written to the screen, type 5 to exit the bank application.

In this example, the input to the bank application is synchronized because at each step, the human user waits for a visual cue before typing.
This cue can be either the appearance of a particular message on the screen, or stability of all the information on the screen.

You can synchronize the input of a Vuser in the same way by using the TE-synchronization functions, **TE_wait_sync**, **TE_wait_text**, **TE_wait_silent**, and **TE_wait_cursor**. These functions effectively emulate a human user who types into a terminal window and then waits for the server to respond, before typing in the next command.

The **TE_wait_sync** function is used to synchronize block-mode (IBM) terminals only. The other TE-synchronization functions are used to synchronize character-mode (VT) terminals.

When you record an RTE Vuser script, VuGen can automatically generate and insert **TE_wait_sync**, **TE_wait_text**, and **TE_wait_cursor** statements into the script. You use VuGen's recording options to specify which synchronization functions VuGen should insert.

Note: Do not include any synchronization statements in the Vuser_end section of a Vuser script. Since a Vuser can be aborted at any time, you cannot predict when the Vuser_end section will be executed.

Synchronizing Block-Mode (IBM) Terminals

The **TE_wait_sync** function is used for synchronization RTE Vusers operating block-mode (IBM) terminals. Block-mode terminals display the "X SYSTEM" message to indicate that the system is in Input Inhibited mode. When a system is in the Input Inhibited mode no typing can take place because the terminal emulator is waiting for a transfer of data from the server.

When you record a script on a block-mode terminal, by default, VuGen generates and inserts a **TE_wait_sync** function into the script each time the "X SYSTEM" message appears. You use VuGen's recording options to specify whether or not VuGen should automatically insert **TE_wait_sync** functions.

When you run a Vuser script, the **TE_wait_sync** function checks if the system is in the X SYSTEM mode. If the system is in the X SYSTEM mode, the **TE_wait_sync** function suspends script execution. When the "X SYSTEM" message is removed from the screen, script execution continues.

Note: You can use the **TE_wait_sync** function only with IBM block-mode terminals emulators (5250 and 3270).

In general, the **TE_wait_sync** function provides adequate synchronization for all block-mode terminal emulators. However, if the **TE_wait_sync** function is ineffective in a particular situation,

you can enhance the synchronization by including a **TE_wait_text** function. For more information on the **TE_wait_text** function, see "Synchronizing Character-Mode (VT) Terminals" on next page, and the Function Reference (**Help > Function Reference**).

In the following script segment, the Vuser logs on with the user name "QUSER" and the password "HPLAB". The Vuser then presses `Enter` to submit the login details to the server. The terminal emulator displays the X SYSTEM message while the system waits for the server to respond.

The TE_wait_sync statement causes the Vuser to wait until the server has responded to the login request, that is, for the X SYSTEM message to be removed—before executing the next line of the script.

```
TE_type("QUSER");
lr_think_time(2);
TE_type("<kTab>HPLAB");
lr_think_time(3);
TE_type("<kEnter>");
TE_wait_sync();
....
```

When a **TE_wait_sync** function suspends the execution of a script while an X SYSTEM message is displayed, the Vuser continues to monitor the system—waiting for the X SYSTEM message to disappear. If the X SYSTEM message does not disappear before the synchronization timeout expires, the **TE_wait_sync** function returns an error code. The default timeout is 60 seconds.

Set the TE_wait_sync synchronization timeout

1. Select **Vuser** > **Run-Time Settings**. The Run-Time Settings dialog box appears.
2. Select the **RTE:RTE** node in the Run-Time setting tree.
3. Under **X SYSTEM Synchronization**, enter a value (in seconds) in the **Timeout** box.
4. Click **OK** to close the Run-Time Settings dialog box.

After a Vuser executes a **TE_wait_sync** function, the Vuser waits until the terminal is no longer in the X S YSTEM mode . When the terminal returns from the X S YSTEM mode, the Vuser continues to monitor the system for a short period to verify that the terminal is fully stable, that is, that the system does not return to the X S YSTEM mode. Only then does the **TE_wait_sync** function terminate and allow the Vuser to continue executing its script. The period that the Vuser continues to monitor the system, after the system has returned from the X SYSTEM mode, is known as the stable time. The default stable time is 1000 milliseconds.

You may need to increase the stable time if your system exhibits the following behavior:

When a system returns from the X SYSTEM mode, some systems "flickers" to and from the X SYSTEM for a short period of time until the system stabilizes. If the system remains out of the X SYSTEM mode for more than one second, and then returns to the X SYSTEM mode, the **TE_wait_sync** function will assume that the system is stable. If a Vuser then tries to type information to the system, the system will shift into keyboard-locked mode.

Alternatively, if your system never flickers when it returns from the X SYSTEM mode, you can reduce the stable time to less than the default value of one second.

Change the stable time for TE_wait_sync functions

1. Select **Vuser** > **Run-Time Settings**. The Run-Time Settings dialog box appears.

2. Select the **RTE:RTE** node.
3. Under **X SYSTEM Synchronization**, enter a value (in milliseconds) in the **Stable time** box.
4. Click **OK** to close the Run-Time Settings dialog box.

   For more information on the **TE_wait_sync** function, see the Function Reference (**Help > Function Reference**).

   You can instruct VuGen to record the time that the system remains in the X SYSTEM mode each time that the X SYSTEM mode is entered. To do so, VuGen inserts a TE_wait_sync_ transaction function after each TE_wait_sync function, as shown in the following script segment:

```
TE_wait_sync();
TE_wait_sync_transaction("syncTrans1");
```

   Each **TE_wait_sync_transaction** function creates a transaction with the name "default." This allows you to analyze how long the terminal emulator waits for responses from the server during a scenario run. You use the recording options to specify whether VuGen should generate and insert **TE_wait_sync_transaction** statements.

Instruct VuGen to insert TE_wait_sync_transaction statements

1. Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.
2. Select the **Generate Automatic X SYSTEM transactions** option, and then click **OK**.

Synchronizing Character-Mode (VT) Terminals

There are three types of synchronization that you can use for character-mode (VT) terminals. The type of synchronization that you select depends on:

- the design of the application that is running in the terminal emulator
- the specific action to be synchronized

Waiting for the Cursor to Appear at a Specific Location

The preferred method of synchronization for VT type terminals is cursor synchronization. Cursor synchronization is particularly useful with full-screen or form-type applications, as opposed to scrolling or TTY-type applications.

Cursor synchronization uses the **TE_wait_cursor** function. When you run an RTE Vuser script, the **TE_wait_cursor** function instructs a Vuser to suspend script execution until the cursor appears at a specified location on the screen. The appearance of the cursor at the specified location means that the application is ready to accept the next input from the terminal emulator.

The syntax of the **TE_wait_cursor** function is:

int TE_wait_cursor (int col, int row, int stable, int timeout);

During script execution, the **TE_wait_cursor** function waits for the cursor to reach the location specified by col , row.

The **stable** parameter specifies the time (in milliseconds) that the cursor must remain at the specified location. If you record a script using VuGen, **stable** is set to 100 milliseconds by default. If the client application does not become stable in the time specified by the **timeout** parameter, the

function returns TIMEOUT. If you record a script using VuGen, **timeout** is set by default to the value of TIMEOUT, which is 90 seconds. You can change the value of both the **stable** and **timeout** parameters by directly editing the recorded script.

The following statement waits for the cursor to remain stable for three seconds. If the cursor doesn't stabilize within 10 seconds, the function returns TIMEOUT.

```
TE_wait_cursor (10, 24, 3000, 10);
```

For more information on the **TE_wait_cursor** function, see the Function Reference (**Help > Function Reference**).

You can instruct VuGen to automatically generate TE_wait_cursor statements, and insert them into a script, while you record the script. The following is an example of a TE_wait_cursor statement that was automatically generated by VuGen:

```
TE_wait_cursor(7, 20, 100, 90);
```

Instruct VuGen to automatically generate TE_wait_cursor statements, and insert them into a script while recording

1. Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.
2. Under **Generate Automatic Synchronization Commands** select the **Cursor** check box, and then click **OK**.

Waiting for Text to Appear on the Screen

You can use text synchronization to synchronize an RTE Vuser running on a VT terminal emulator. Text synchronization uses the **TE_wait_text** function. During script execution, the **TE_wait_text** function suspends script execution and waits for a specific string to appear in the terminal window before continuing with script execution. Text synchronization is useful with those applications in which the cursor does not consistently appear in a predefined area on the screen.

Note: Although text synchronization is designed to be used with character mode (VT) terminals, it can also be used with IBM block-mode terminals. Do not use automatic text synchronization with block-mode terminals.

The syntax of the **TE_wait_text** function is:

```
int TE_wait_text (char * pattern, int timeout, int col1, int row1, int col2, int row2,
        int * retcol, int * retrow, char * match );
```

This function waits for text matching pattern to appear within the rectangle defined by col1, row1, col2, row2. Text matching the pattern is returned to **match**, and the actual row and column position is returned to **retcol** and **retrow**. If the **pattern** does not appear before the **timeout** expires, the function returns an error code. The **pattern** can include a regular expression. See the Function Reference for details on using regular expressions. Besides the **pattern** and **timeout** parameters, all the other parameters are optional.

If **pattern** is passed as an empty string, the function will wait for timeout if it finds any text at all within the rectangle. If there is no text, it returns immediately.

If the pattern does appear, then the function waits for the emulator to be stable (finish redrawing, and not display any new characters) for the interval defined by the TE_SILENT_SEC and TE_

SILENT_MILLI system variables. This, in effect, allows the terminal to become stable and emulates a human user.

If the terminal does not become stable within the interval defined by TE_SILENT_TIMEOUT, script execution continues. The function returns 0 for success, but sets the TE_errno variable to indicate that the terminal was not silent after the text appeared.

To modify or retrieve the value of any of the TE_SILENT system variables, use the TE_getvar and TE_setvar functions. For more information, see the Function Reference (**Help > Function Reference**).

In the following example, the Vuser types in its name, and then waits for the application to respond.

```
/* Declare variables for TE_wait_text */
int ret_row;
int ret_col;
char ret_text [80];
/* Type in user name. */
TE_type ("John");
/* Wait for teller to respond. */
TE_wait_text ("Enter secret code:", 30, 29, 13, 1, 13, =;ret_col,
=;ret_row,
            ret_text);
```

You can instruct VuGen to automatically generate **TE_wait_text** statements, and insert them into a script, while you record the script.

Instruct VuGen to automatically generate TE_wait_text statements, and insert them into a script while recording

1. Select **Vuser** > **Recording Options**. The Recording Settings dialog box appears.
2. Under **Generate Automatic Synchronization Commands**, select the **Prompt** check box, and then click **OK**.

   The following is an example of a TE_wait_text statement that was automatically generated by VuGen. The function waits up to 20 seconds for the string "keys" to appear anywhere on the screen. Note that VuGen omits all the optional parameters when it generates a TE_wait_text function.

   ```
   TE_wait_text("keys", 20);
   ```

Waiting for the Terminal to be Silent

In instances when neither cursor synchronization nor text synchronization are effective, you can use "silent synchronization" to synchronize the script. With "silent synchronization," the Vuser waits for the terminal emulator to be silent for a specified period of time. The emulator is considered to be silent when it does not receive any input from the server for a specified period of time.

Note: Use silent synchronization only when neither cursor synchronization nor text synchronization are effective.

You use the **TE_wait_silent** function to instruct a script to wait for the terminal to be silent. You specify the period for which the terminal must be silent. If the terminal is silent for the specified

period, then the **TE_wait_silent** function assumes that the application has stopped printing text to the terminal screen, and that the screen has stabilized.

The syntax of the function is:

```
int  TE_wait_silent (int  sec, int  milli, int  timeout );
```

The **TE_wait_silent** function waits for the terminal emulator to be silent for the time specified by sec (seconds) and milli (milliseconds). The emulator is considered silent when it does not receive any input from the server. If the emulator does not become silent (i.e. stop receiving characters) during the time specified by the time timeout variable, then the function returns an error.

For example, the following statement waits for the screen to be stable for three seconds. If after ten seconds, the screen has not become stable, the function returns an error.

```
TE_wait_silent (3, 0, 10);
```

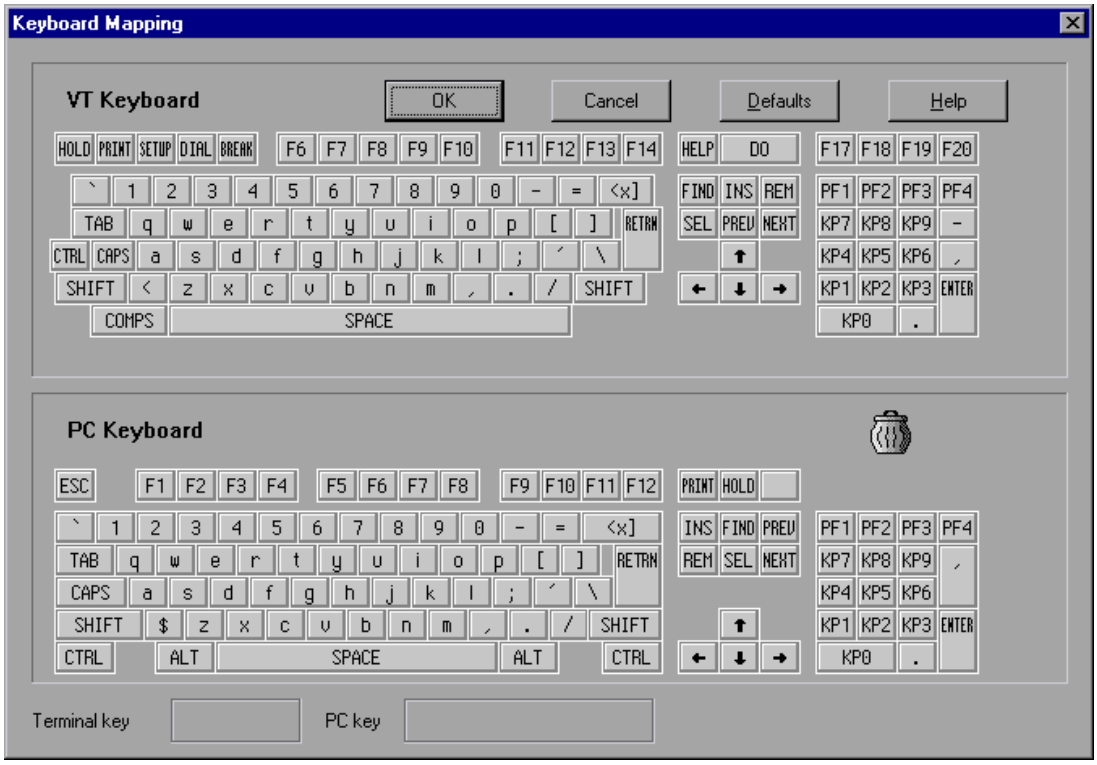
For more information, see the Function Reference (**Help > Function Reference**).

How to Map Terminal Keys to PC Keyboard Keys

Because you are using a terminal emulator, you will be using a PC keyboard in place of a terminal keyboard. Many keys that are found on the terminal keyboard are not available on a PC keyboard. Examples of such keys are HELP, AUTHOR, and PUSH, which are found on the IBM 5250 keyboard. To successfully operate the terminal emulator and any associated application programs, you may have to map certain terminal keys to keys on the PC keyboard.

Map a Terminal Key to a Key on the PC Keyboard

1. In the terminal emulator, select **Options** > **Keyboard Map**, or click the **Keyboard Mapping** button  . The Keyboard Mapping dialog box opens.



2. Click the Keyboard **Mapping** button on the toolbar. To map a terminal key to a PC key, drag a key from the upper terminal keyboard to a PC key on the lower keyboard.

   You can click the Shift and/or Control keys on the upper keyboard to display additional key functions that can be viewed only by first selecting either of these keys. You can then drag the required key from the upper terminal keyboard to a key on the lower PC keyboard.

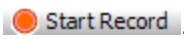
   To cancel a definition, drag the PC key definition to the wastebasket. This restores the default function of the PC key.

   To restore the default mappings, click **Defaults**.

How to Record RTE Vuser Scripts

You use VuGen to record RTE Vuser scripts. VuGen uses the PowerTerm terminal emulator to emulate a wide variety of terminal types.

This task describes how to record RTE Vuser scripts. This procedure differs from the general recording procedure in "Recording" on page 106.

1. **Record the terminal setup and connection**
   a. Open an existing RTE Vuser script, or create a new one.
   b. In the **Sections** box, select the **vuser_init** section to insert the recorded statements.
   c. In the Vuser script, place the cursor at the location where you want to begin recording.
   d. Click the **Start Record** button 

. The PowerTerm main window opens.

e. From the PowerTerm menu bar, select **Terminal** > **Setup** to display the Terminal Setup dialog box.

f. Select the type of emulation from the VT Terminal and IBM Terminal types, and then click **OK**.

> **Note:** Select an IBM terminal type to connect to an AS/400 machine or an IBM mainframe; select a VT terminal type to connect to a Linux workstation.

g. Select **Communication** > **Connect** to display the Connect dialog box.

h. Under **Session Type**, select the type of communication to use.

i. Under **Parameters**, specify the required options. The available parameters vary depending on the type of session that you select. For details on the parameters, click **Help**.

> **Tip:** Click **Save As** to save the parameter-sets for re-use in the future. The parameter-sets that you save are displayed in the Sessions List box.

j. Click **Connect**. PowerTerm connects to the specified system, and VuGen inserts a **TE_connect** function into the script, at the insertion point. The **TE_connect** statement has the following form:

```
/* *** The terminal type is VT 100. */
TE_connect(
    "comm-type = telnet;"
    "host-name = alfa;"
    "telnet-port = 992;"
    "terminal-id = ;"
    "set-window-size = true;"
    "security-type = ssl;"
    "ssl-type = tls1;"
    "terminal-type = vt100;"
    "terminal-model = vt100;"
    "login-command-file = ;"
    "terminal-setup-file = ;"
    , 60000);
if (TE_errno != TE_SUCCESS)
    return -1;
```

The inserted **TE_connect** statement is followed by an if statement that checks whether or not the **TE_connect** function succeeds during replay**.**

> **Note:** Do not record more than one connection to a server **(TE_connect)** in a Vuser script.

2. Record typical user actions

After recording the setup procedure, you perform typical user actions or business processes. You record these processes into the **Actions** section of the Vuser script. Only the **Actions** section of a Vuser script is repeated when you run multiple iterations of the script.

When recording a session, VuGen records the text strokes and not the text. Therefore, it is not recommended that you copy and paste commands into the PowerTerm window—instead, type them in directly.

a. Select the **Actions** section in the **Section** box.

b. Proceed to perform typical user actions in the terminal emulator. VuGen generates the appropriate statements, and inserts them into the Vuser script while you type. If necessary, you can edit the recorded statements while you record the script.

3. **Record the log-off procedure**

a. Make sure that you have performed and recorded the typical user actions as described in the previous section.

b. In the VuGen main window, click **vuser_end** in the **Section** box.

c. Perform the log off procedure. VuGen records the procedure into the **vuser_end** section of the script.

d. Click **Stop Recording** on the Recording toolbar. The main VuGen window displays all the recorded statements.

e. Click **Save** to save the recorded session. After recording a script, you can manually edit it in VuGen's main window.

How to Implement Continue on Error

To configure the Continue on Error functionality in RTE Scripts:

- To continue running the script on error, insert the following function:

TE_setvar(TE_IGNORE_ERRORS, 1)

- To restore the default behavior of failing the script on error, insert the following function:

TE_setvar(TE_IGNORE_ERRORS, 0)

SAP Protocols

Selecting a SAP Protocol Type

- To test the SAP GUI user operating only on the client, use the SAP GUI Vuser type.
- To test a SAP GUI user that also uses a Web browser, use the SAP (Click & Script) or SAP-Web protocol.

To record a SAP GUI session that uses browser controls, create a multi-protocol Vuser script with the SAP GUI and SAP-Web protocols. This allows VuGen to record Web-specific functions when encountering the browser controls. This will not work if you attempt to combine SAP GUI and Web protocols.

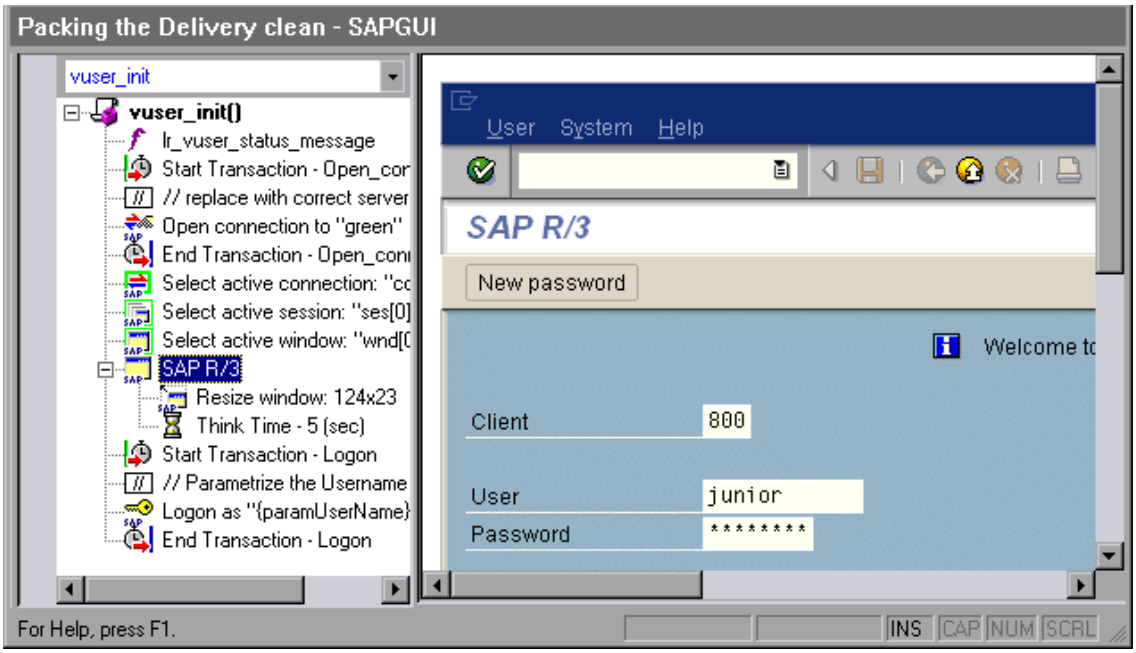
Before recording a session, verify that your modules and client interfaces are supported by VuGen. The following table describes the SAP client modules for SAP Business applications and the relevant tools:

SAP module	VuGen support
SAP Web Client or mySAP.com	Use the SAP-Web protocol.
SAP GUI for Windows	Use the SAP GUI protocol. This also supports APO module recording (requires patch level 24 for APO 3.0 for SAP 6.20).
SAP GUI for Windows and a web browser	Use the SAP (Click & Script) protocol.
SAP GUI for Java	This client is not supported.

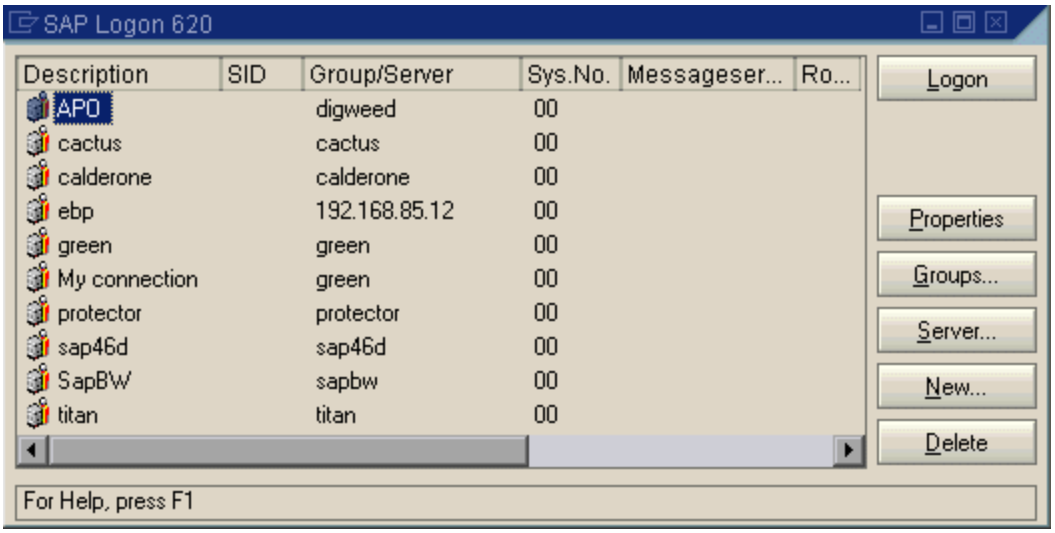
SAP GUI Protocol

The SAP GUI Vuser script typically contains several SAP transactions which make up a business process. A business process consists of functions that emulate user actions. Open the **Step Navigator** to see each user action as a Vuser script step.

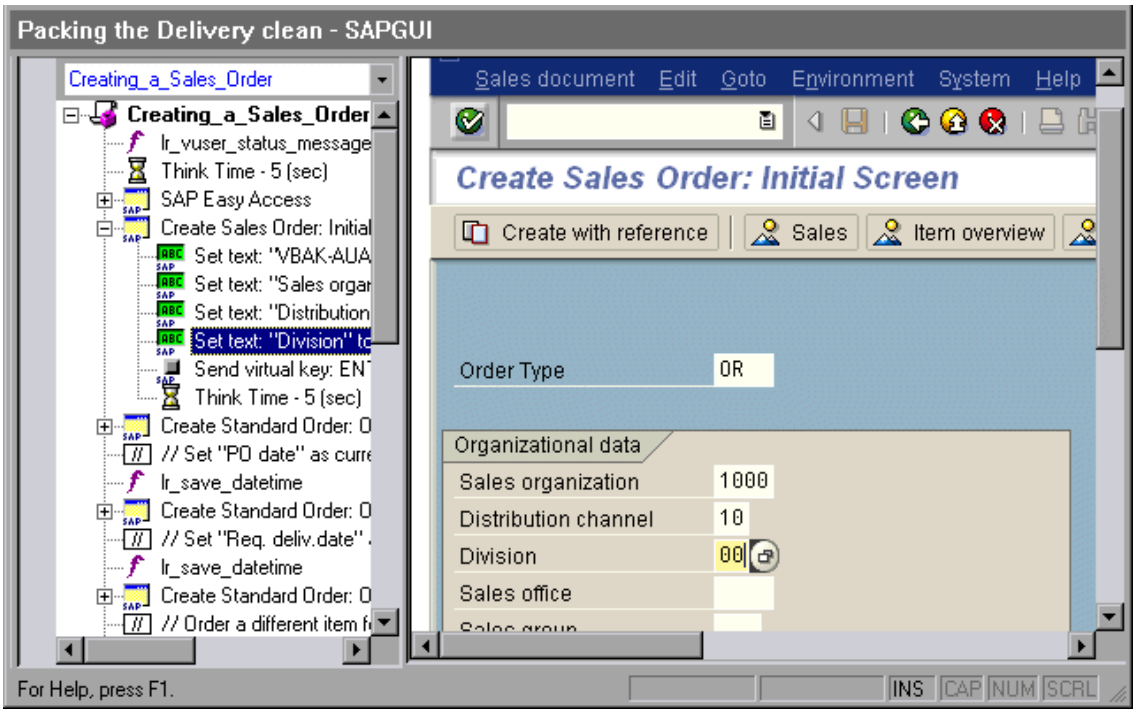
The following example shows a typical recording of a SAP GUI client. The first section, **vuser_init**, contains the opening of a connection and a logon.



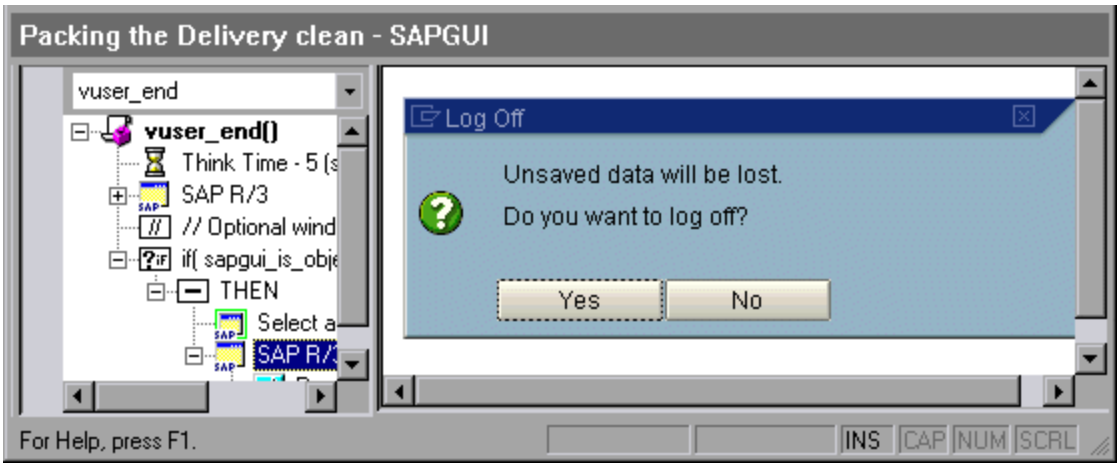
Note that the Open Connection step uses one of the connection names in the SAP Logon **Descriptions** list. If the specified connection name is not in the list, the Vuser looks for a server with that name.



In the following section, the functions emulate typical user operations such as menu selection and the setting of a check box.



The final section, **vuser_end**, illustrates the logoff procedure.



When recording a multi- protocol script for both SAP GUI and Web, VuGen generates steps for both protocols. In the Script view, you can view both **sapgui** and **web** functions.

The following example illustrates a multi-protocol recording in which the SAP GUI client opens a Web control. Note the switch from **sapgui** to **web** functions.

```
sapgui_tree_double_click_item("Use as general WWW browser, REPTITLE",
    "shellcont/shell",
    "000732",
    "REPTITLE",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1020",
    END_OPTIONAL);
...
sapgui_set_text("",
    "http:\\\\yahoo.com",
    "usr/txtEDURL",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1021",
    END_OPTIONAL);
...
web_add_cookie("B=7pt5cisv1p3m2=;b=2; DOMAIN=www.yahoo.com");
web_url("yahoo.com",
        "URL=http://yahoo.com/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        EXTRARES,

"URL=http://srd.yahoo.com/hpt1/ni=17/ct=lan/sss=1043752588/t1=10437525-
75385/d1=1251
          /d2=1312/d3=1642/d4=4757/0.4097009487287739/*1",
          "Referer=http://www.yahoo.com/", ENDITEM,
        LAST);
```

SAP Web Protocol

The SAP-Web Vuser script typically contains several SAP transactions which make up a business process. The business process consists of functions that emulate user actions. For information about these functions, see the Web functions in the Function Reference (**Help > Function Reference**).

Example:

The following example shows a typical recording for an SAP Portal client:

```
vuser_init()
{
    web_reg_find("Text=SAP Portals Enterprise Portal 5.0",
        LAST);
    web_set_user("junior{UserNumber}",
        lr_decrypt("3ed4cfe457afe04e"),
        "sonata.hplab.com:80");
    web_url("sapportal",
        "URL=http://sonata.hplab.com/sapportal",
        "Resource=0",
        "RecContentType=text/html",
        "Snapshot=t1.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=/SAPPortal/IE/Media/sap_mango_
polarwind/images/header/branding_image.jpg",

"Ref-
erer=http:-
//sonata.hplab.com/hrnp$30001/sonata.hplab.coml:80/Action/26011
[header]"
          , ENDITEM,
        "Url=/SAPPortal/IE/Media/sap_mango_
polarwind/images/header/logo.gif",

"Ref-
erer=http:-
//sonata.hplab.com/hrnp$30001/sonata.hplab.com:80/Action/26011[header]
",
            ENDITEM,
...
        LAST);
```

The following section illustrates an SAP Web and SAP GUI multi-protocol recording in which the Portal client opens an SAP control. Note the switch from **web_xxx** to **sapgui_xxx** functions.

```
web_url("dummy",
        "URL=http://sonata.hplab.com:1000/hrnp$30000/sonata.hplab.com:
          1000/Action/dummy?PASS_PARAMS=YES=;dummyComp=dummy=;
Tcode=VA01=;draggable=0=;CompFName=VA01=;Style=sap_mango_polarwind",
```

```
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://sonata.hplab.com/sapportal",
        "Snapshot=t9.inf",
        "Mode=HTML",
        LAST);
    sapgui_open_connection_ex(" /H/Protector/S/3200 /WP",
        "",
        "con[0]");
    sapgui_select_active_connection("con[0]");
    sapgui_select_active_session("ses[0]");
    /*Before running script, enter password in place of asterisks in
logon function*/
    sapgui_logon("JUNIOR{UserNumber}",
        "ides",
        "800",
        "EN",
        BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui102",
        END_OPTIONAL);
```

SAP (Click & Script) Protocol

VuGen can create Vuser scripts for SAP Enterprise portal7 and SAP ITS 6.20/6.40 environments using specialized test objects and methods that have been customized for SAP. The objects are APIs based on HP QuickTest or Unified Functional Testing support for SAP.

As you record a test or component on your SAP application, VuGen records the operations you perform. VuGen recognizes special SAP Windows objects such as frames, table controls, iViews, and portals.

VuGen supports recording for the following SAP controls: button, checkbox, drop-down menu, edit field, iview, list, menu, navigation bar, OK code, portal, radio group, status bar, tab strip, table, and tree view.

VuGen uses the control handler layer to create the effect of an operation on a GUI control. During recording, when encountering one of the supported SAP objects, VuGen generates a function with an **sap_xxx** prefix.

Example:

In the following example, a user selected the **User Profile** tab. VuGen generated a **sap_portal** function.

```
web_browser("Close_2",
        "Snapshot=t7.inf",
        DESCRIPTION,
        "Ordinal=2",
        ACTION,
        "UserAction=Close",
        LAST);
lr_think_time(7);
```

```
web_text_link("Personalize",
        "Snapshot=t8.inf",
        DESCRIPTION,
        "Text=Personalize",
        ACTION,
        "UserAction=Click",
        LAST);
    lr_think_time(6);
sap_portal("Sap Portal_2",
        "Snapshot=t9.inf",
        DESCRIPTION,
        "BrowserOrdinal=2",
        ACTION,
        "DetailedNavigation=User Profile",
        LAST);
```

Note: When you record an SAP (Click & Script) session, VuGen generates standard Web (Click & Script) functions for objects that are not SAP-specific. You do not need to explicitly specify the Web protocol. In the example above, VuGen generated a **web_text_link** function when the user clicked the **Personalize** button.

Replaying SAP GUI Optional Windows

When working with SAP GUI Vuser Scripts, you may encounter optional windows in the SAP GUI client—windows that were present during recording, but do not exist during replay. If you try to replay your recorded script as is, it will fail when it attempts to find the missing windows.

VuGen's optional window mechanism performs the actions on a window only after verifying that it exists. The Vuser checks if the window indicated in the **Select active window** step exists. If the window is found during replay, it performs the actions as they were recorded in the script. If it does not exist, the Vuser ignores all window actions until the next **Select active window** step. Note that only SAP GUI steps (beginning with a **sapgui** prefix) are ignored.

To use this feature, in Tree view select the appropriate Select Active Window step and select **Run steps for window only if it exists** from the right-click menu.

To disable this feature and attempt to run these steps at all times, regardless of whether the Vuser finds the window or not, select **Always run steps for this window** from the right-click menu.

How to Configure the SAP Environment

This task describes configure and verify the SAP environment for use with VuGen.

VuGen support for the SAP GUI for Windows client, is based on SAP's Scripting API. This API allows Vusers to interact with the SAP GUI client, receive notifications, and perform operations.

The Scripting API is available only in recent versions of the SAP Kernel. In kernel versions that support scripting, the option is disabled by default. In order to use VuGen, first make sure that the SAP servers support the Scripting API, and enable the API on both the server and clients. For more information and to download patches, see the SAP OSS note #480149.

VuGen provides a utility that checks if your system supports scripting. The utility, **VerifyScript.exe**, is located on DVD in the **Additional Components\SAP_Tools\VerifySAPGUI** folder. For more information, see the file **VerifyScripting.htm** provided with this utility.

Checking the SAP GUI for Windows Client Patch Level

You can check the patch level of your SAP GUI for Windows client from the About box. The lowest patch level supported is version 6.20 patch 32.

Check the Patch Level

1. Open the SAP GUI logon window. Click the top left corner of the SAP Logon dialog box and select **About SAP Logon** from the menu.
2. The SAP version information dialog box opens. Verify that the Patch Level entry is 32 or higher.

Check the Kernel Patch Level

1. Log in to the SAP system
2. Select **System > Status**
3. Click the **Other kernel information**  button.
4. In the **Kernel Information** section, check the value of the **Sup. Pkg. lvl.**

   The level must be greater than the level listed in the following chart depending on the SAP version you are using.

Software Component	SAP Release	Kernel Patch Level
SAP_APPL	31I	Kernel 3.1I level 650
SAP_APPL	40B	Kernel 4.0B level 903
SAP_APPL	45B	Kernel 4.5B level 753
SAP_BASIS	46B	Kernel 4.6D level 948
SAP_BASIS	46C	Kernel 4.6D level 948
SAP_BASIS	46D	Kernel 4.6D level 948
SAP_BASIS	610	Kernel 6.10 level 360

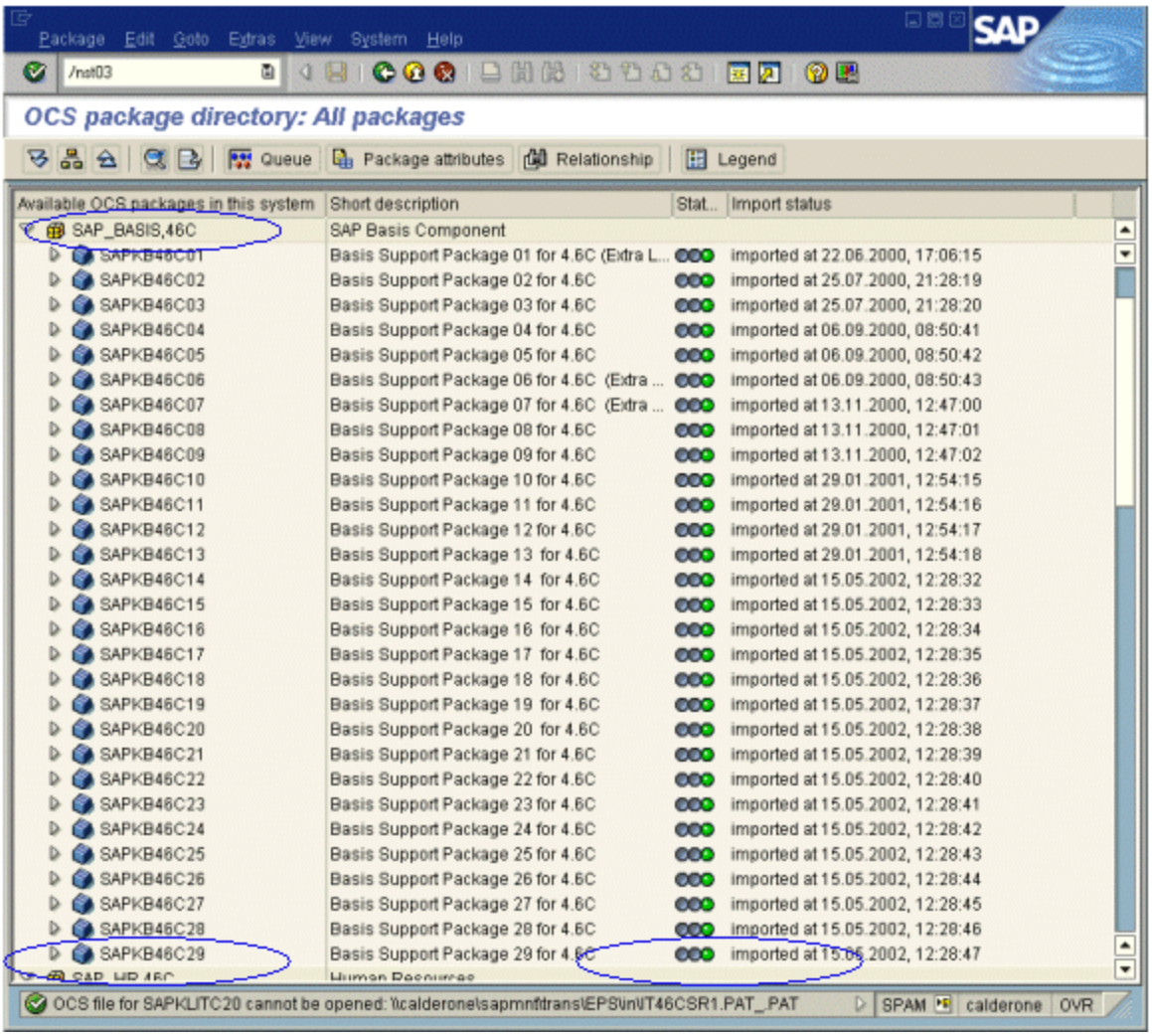
Check the R/3 Support Packages

1. Log on to the SAP system and run the SPAM transaction.
2. In the **Directory** section, select **All Support Packages**, and click the **Display** button.
3. Verify that the correct package is installed for your version of SAP according to the table below.

Software Component	Release	Package Name
SAP_APPL	31I	SAPKH31I96

SAP_APPL	40B	SAPKH40B71
SAP_APPL	45B	SAPKH45B49
SAP_BASIS	46B	SAPKB46B37
SAP_BASIS	46C	SAPKB46C29
SAP_BASIS	46D	SAPKB46D17
SAP_BASIS	610	SAPKB61012

If the correct version is installed, a green circle appears in the Status column.



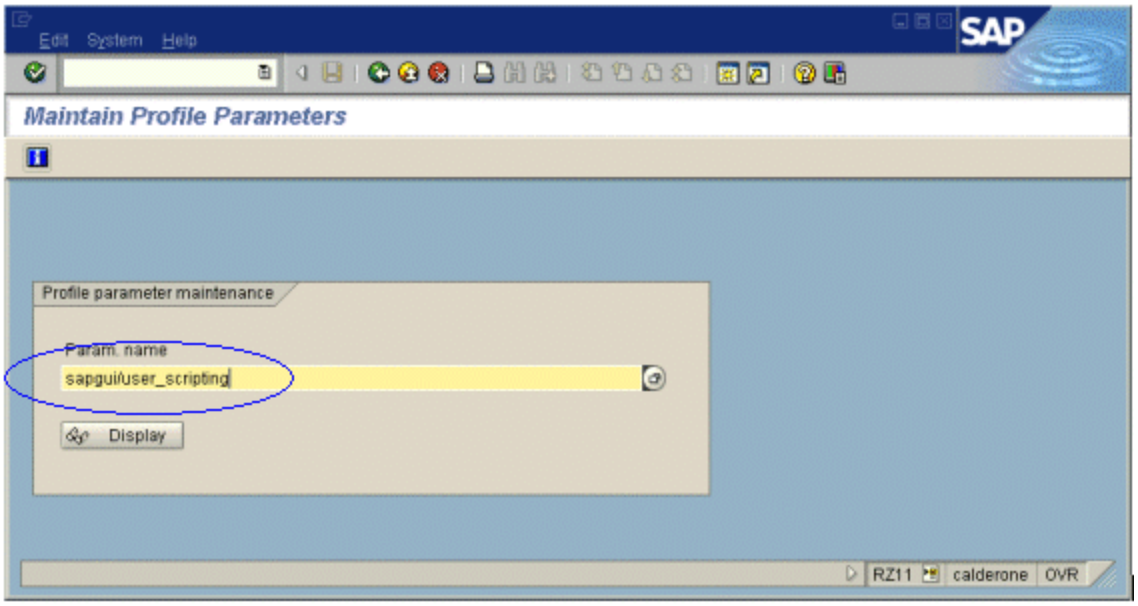
If you do not have the OCS package installed, download it the from the www.sap.com Web site and install it. For more information, see the SAP OSS note #480149.

Enable scripting on the SAP Application Server

A user with administrative permissions enables scripting by setting the **sapgui/user_scripting** profile parameter to **TRUE** on the application server. To enable scripting for all users, set this

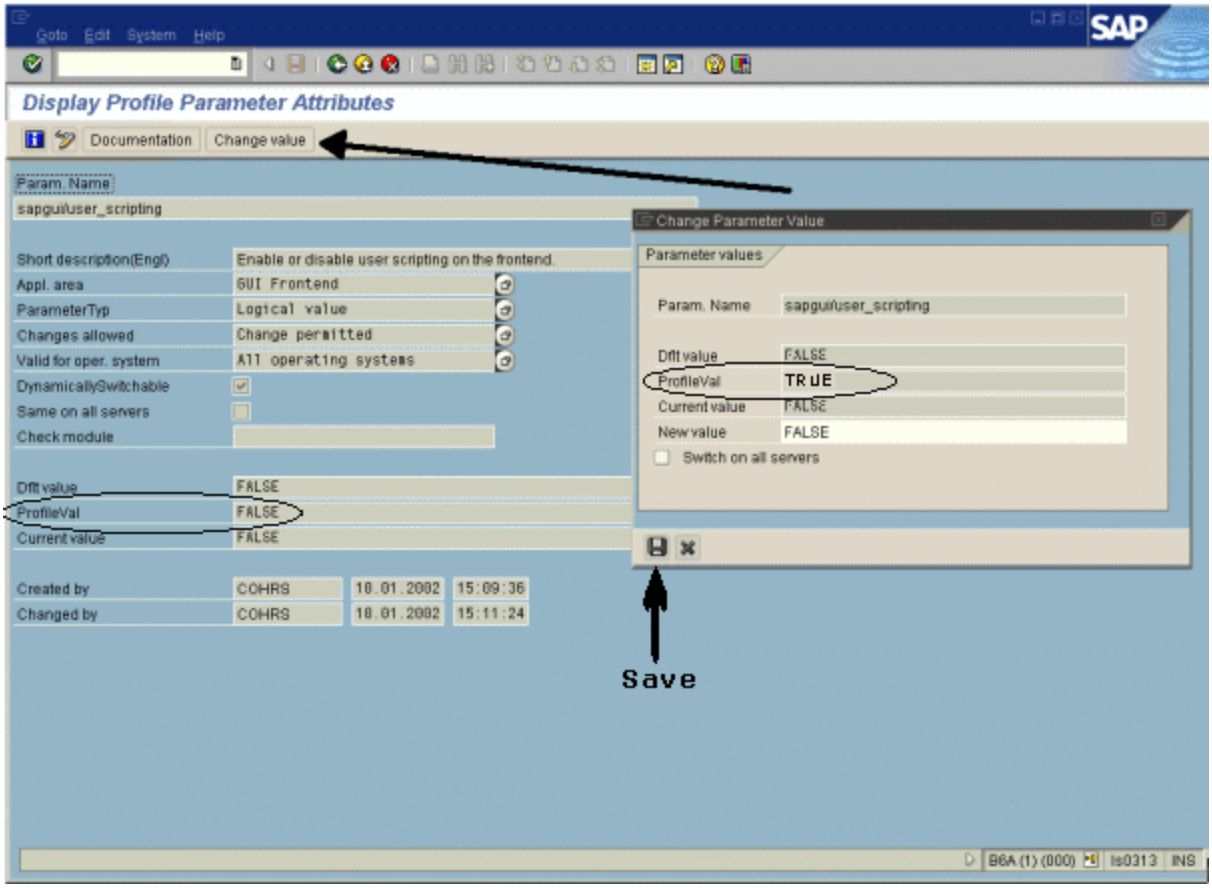
parameter on all application servers. To enable scripting for a specific group of users, only set the parameter on application servers with the desired access restrictions. The following steps describe how to change the profile parameter.

1. Open transaction **rz11.** Specify the parameter name **sapgui/user_scripting** and click **Display**. The Display Profile Parameter Attributes window opens.



If **Parameter name is unknown** appears in the status bar, this indicates that you are missing the current Support Package. Import the Support Package that corresponds to the SAP BASIS and kernel versions of the application server, as described in the steps above.

2. If **Profile Val** is FALSE, you need to modify its value. Click the **Change value** button in the toolbar. The Change Parameter Value window opens. Enter TRUE in the **ProfileVal** box and click the **Save** button.



When you save the change, the window closes and **ProfileVal** is set to TRUE.

3. Restart the application server, since this change only takes effect when you log onto the system.

If the updated **ProfileVal** did not change, even after restarting the server, then the kernel of the application server is outdated. Import the required kernel patch, as specified in the steps above.

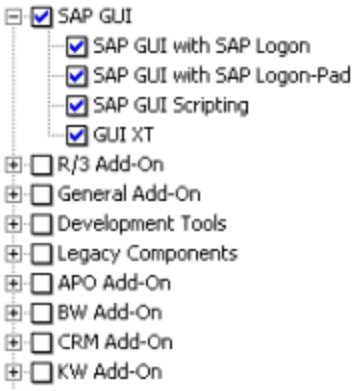
Note that the Profile Value may be dynamically activated in the following kernel versions, using transaction rz11, without having to restart the application server.

Release	Kernel Version	Patch Level
4.6B, 4.6C, 4.6D	4.6D	972
6.10	6.10	391
6.20	all versions	all levels

Enable scripting on SAP GUI 6.20 client

To allow VuGen to run scripts, you must also enable scripting on the SAP GUI client. You should also configure the client not to display certain messages, such as when a connection is established, or when a script is attached to the GUI process. The following steps describe how to configure the SAP GUI client to work with .

1. **During installation.** While installing the SAP GUI client, enable the **SAP GUI Scripting** option.



2. **After installation.** Suppress warning messages. Open the Options dialog box in the SAP GUI client. Select the **Scripting** tab and clear the following options:
   - **Notify when a script attaches to a running GUI**
   - **Notify when a script opens a connection**

   You can also prevent these messages from popping up by setting the values **WarnOnAttach** and **WarnOnConnection** in the following registry key to 0:

   `HKCU\SOFTWARE\SAP\SAPGUI Front\SAP Frontend Server\Security.`

How to Record SAP GUI Scripts

The following steps describe some prerequisites to recording a SAP GUI script.

Close SAPLogon Application When Recording with Multi

When recording a multi-protocol script in which the SAP GUI client contains Web controls, close the SAPLogon application before recording.

Use Modal Dialog Boxes for F1 Help

Instruct the SAP GUI client to open the F1 help in a modal dialog box as follows:

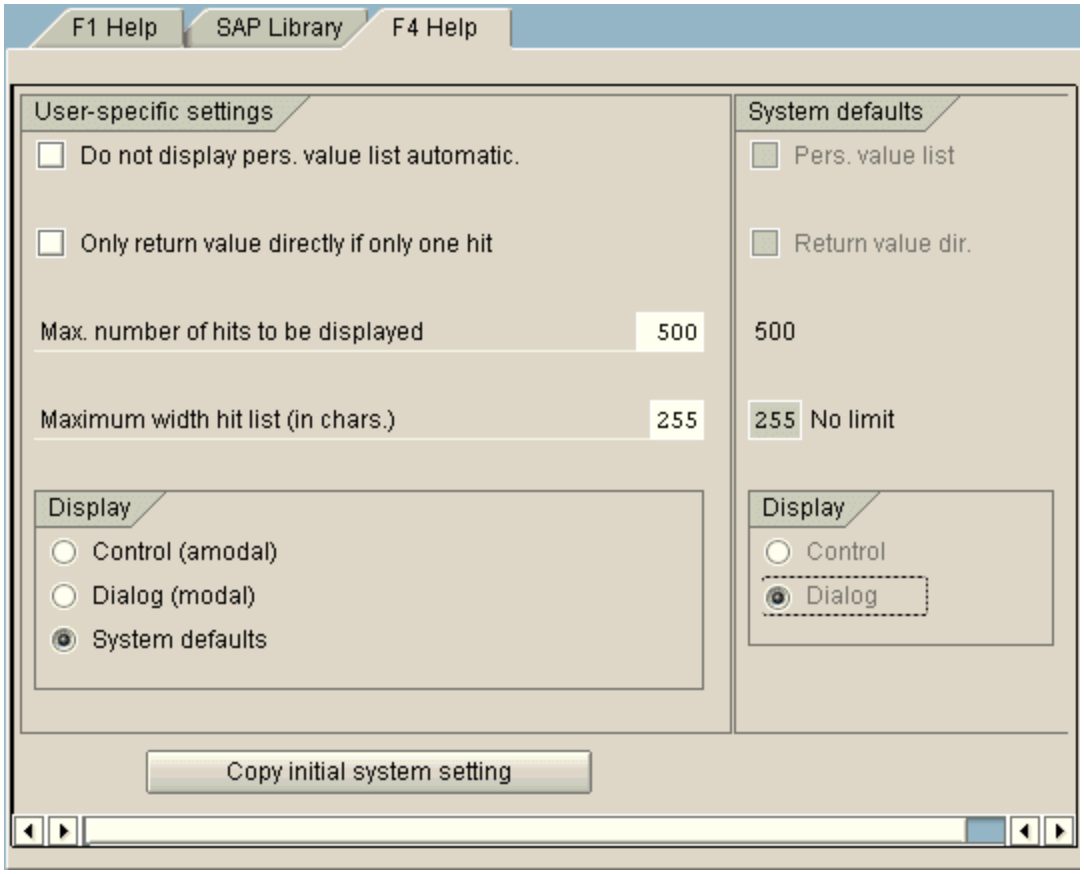
1. Select **Help > Settings**.
2. Click the **F1 Help** tab.
3. Select **in modal dialog box** in the Display section.

Use Modal Dialog boxes for F4 Help

Note: This procedure can only be performed by the administrator.

Instruct the SAP GUI client to open the F4 help in a modal dialog box:

1. Make sure that all users have logged off from the server.
2. Select **Help > Settings**. Click the **F4 Help** tab.



3. In the Display section, select **System defaults**.
4. In the Display portion of the System defaults section, select **Dialog**.
5. Save the changes by clicking **Copy initial system setting**.
6. Verify that the status bar displays the message **Data was saved**.
7. Close the session and restart the service through the SAP Management Console.

How to Replay SAP GUI Scripts

The following steps describe prerequisites to replaying SAPGUI scripts.

Replace Encrypted Password

Replace the encrypted password in the **sapgui_logon** function generated during recording, with the real password. It is the second argument of the function, after the following user name

```
sapgui_logon("user", "pswd", "800", "EN");
```

For additional security, you can encrypt the password within the code. Select the password text (the actual text, not *****) and select **Encrypt string** from the right-click menu. VuGen inserts an **lr_decrypt** function at the location of the password as follows:

```
sapgui_logon("user", lr_decrypt("3ea037b758"), "800", "EN");
```

Display SAP GUI During Replay (optional)

When running a script for the first time, configure VuGen to show the SAP GUI user interface during replay, in order to see the operations being performed through the UI. Select **Replay > Run-Time Settings > SAPGUI > General** node and select **Show SAP Client During Replay**. During a load scenario, disable this option, since it uses a large amount of system resources in displaying the UI for multiple Vusers.

How to Run SAP GUI Scripts in a Scenario

The following steps describe tips for running SAP GUI scripts in a scenario.

LoadRunner Controller Settings

When working with a LoadRunner scenario, set the following values when running your script in a load test configuration:

- **Ramp-up.** One by one (to insure proper logon) in the Scheduler.
- **Think time.** Random think time in the run-time settings.
- **Users per load generator.** 50 Vusers for machine with 512 MB of memory in the Load Generators dialog box.

Make Sure the Agent is Running in Process Mode

Make sure that the LoadRunner (or Performance Center) Remote Agent is running in Process mode. Service mode is not supported.

To check this, move your mouse over the agent's icon in the Windows task bar area, and read the description. If the description reads LoadRunner Agent Service, it is running as a service.

The following steps describe how to restart the agent as a process.

1.  Stop the agent. Right-click the LoadRunner Agent icon and select **Close**.
2. Run **magentproc.exe**, located in the **launch_service\bin** folder, under the LoadRunner installation.
3. To make sure that the correct Agent is launched the next time you start your machine, change the Start type of the Agent Service from Automatic to Manual. Then add a shortcut to **magentproc.exe** to the Windows Startup folder.

   - **Terminal Sessions.** Machines running SAP GUI Vusers may be limited in the number of Vusers that can run, due to the graphic resources available to that machine. To increase the number of Vusers per machine, open additional terminal server sessions on the Load Generator machines. Select **Agent Configuration** from **Start > All Programs > <product_name> > Advanced Settings**, and select the **Enable Terminal Service** option. You specify the number of terminal sessions in the Load generator machine properties. For more information, see Configuring Terminal Services in the .

     Note: When the LoadRunner Agent is running in a terminal session, and the terminal

session's window is minimized, no snapshots will be captured on errors.

How to Enhance SAP GUI Scripts

The following steps describe addition options that are available to enhance SAP GUI scripts.

Record at the Cursor

VuGen allows you to record actions into an existing script by either inserting new actions or replacing existing actions. You may decide to record into an existing script for several reasons:

- You made a mistake in the actions that you performed during recording.
- Your actions were correct, but you need to add additional information such as the handling of popup windows. For example the SAP server may issue an inventory warning, which did not apply during the recording session.

The following steps describe how to record at the cursor.

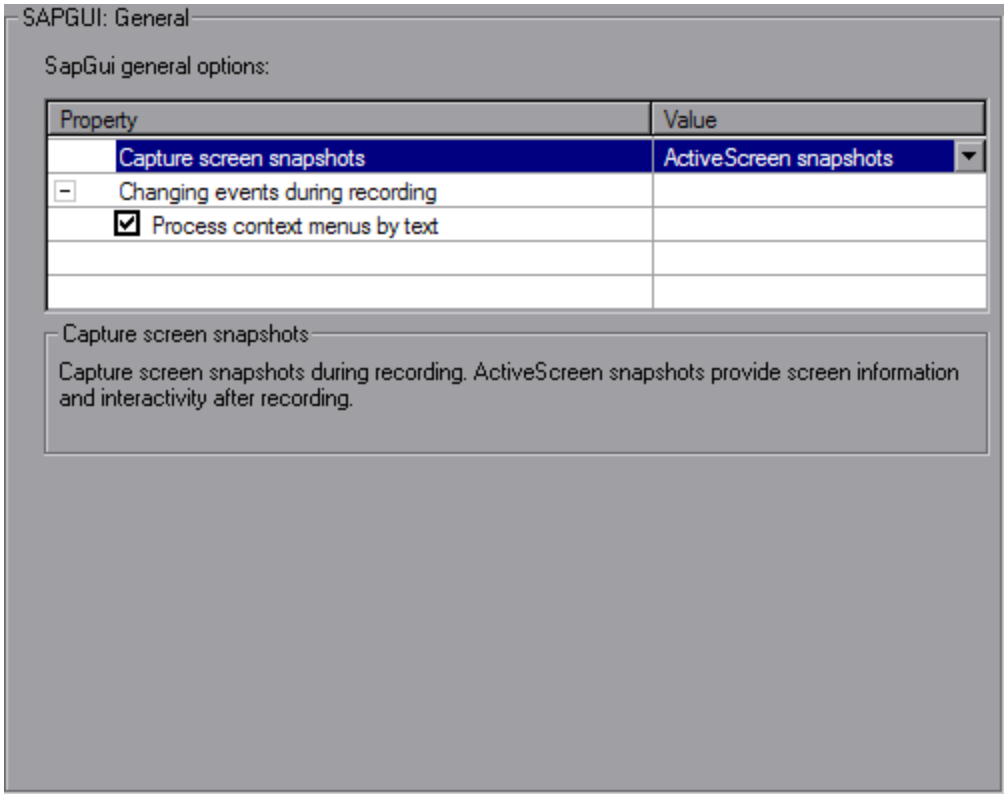
1. Open the Vuser script in the **Editor** and click in the left margin adjacent to an existing function.
2. Click the **Recording at the Cursor** button. VuGen prompts you to make a selection.
3. Select **Insert steps into action** or **Overwrite the rest of the script**.
   a. **Insert steps into action** inserts the newly recorded steps at the cursor without overwriting any existing steps. The new segment is enclosed with comments indicating the beginning and end of the added section. This option is ideal for handling occasional popup windows that were not present during the recording.
   b. **Overwrite the rest of the script** replaces all steps from the point of the cursor onward. This option overwrites the remainder of the current Action and deletes all other Actions. It does not affect the **vuser_init** or **vuser_end** sections.
4. Click **OK**. VuGen replays the script until the point of the cursor.
5. Wait for the floating recording toolbar to open.
6. Perform the required actions in the SAP GUI client, switching between sections and actions as required.
7. To end the recording session, click the **Stop** button on the floating toolbar.

Insert Steps Interactively into a SAP GUI Script

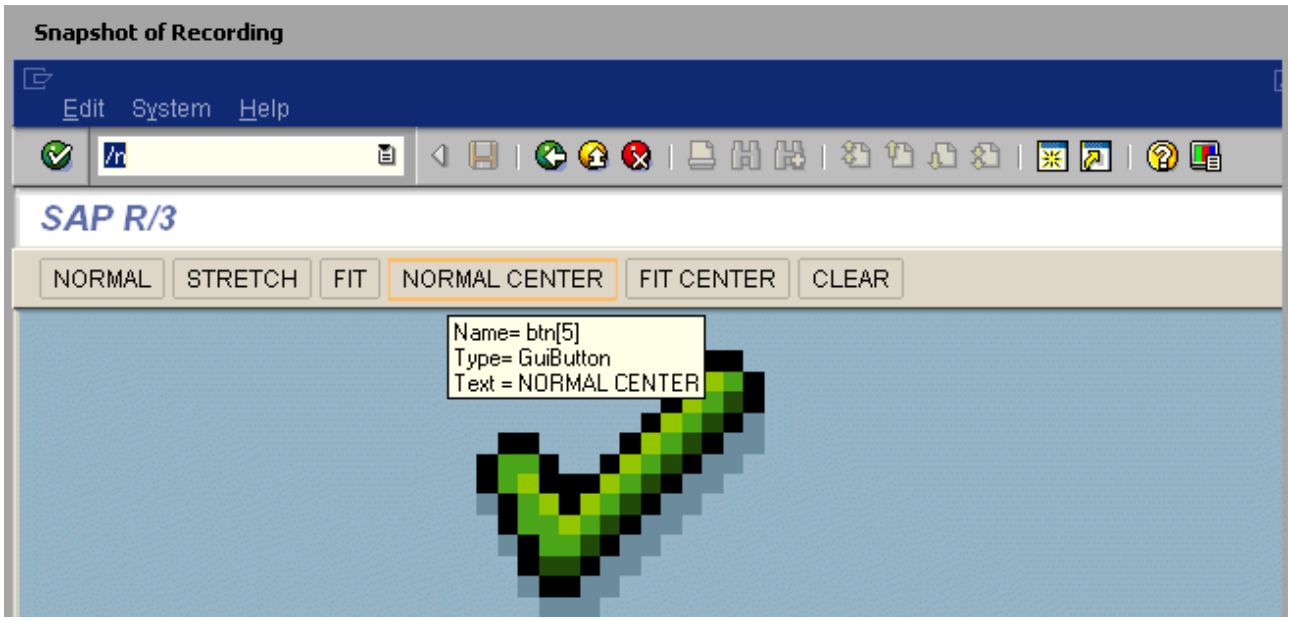
After recording, you can manually add steps to the script in either the **Editor** or **Step Navigator**. In addition to manually adding new functions, you can add new steps interactively for SAP GUI Vusers, directly from the snapshot. Using the right-click menu, you can add object-related steps.

When adding a step from within a snapshot, VuGen uses the Active Screen capability and determines the ID of each object in the SAP GUI client window (unless you disabled Active Screen snapshots in the "SAPGUI > General Node" on page 326). The following steps describe how to insert a step interactively for a specific object.

1. Verify that you recorded the script when Active Screen snapshots were selected in the SAPGUI General node of the Recording Options (enabled by default).

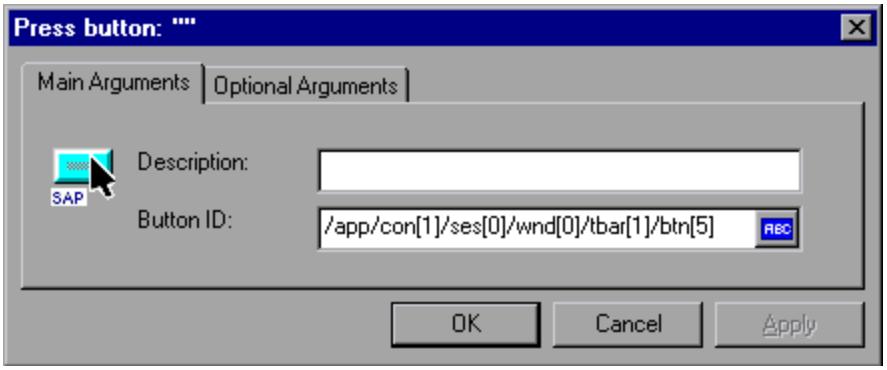


2. Click within the Snapshot pane.
3. Move the mouse over the object for which you want to add a function. Make sure that VuGen recognizes the object and encloses it with a box.



4. Right-click the object, click **Insert New Step**, and then select a step from the list of steps that are available for the object.

The step's Properties dialog box opens, with the Control ID of the object when relevant. For example, if you add a **Press Button** step, for the normal center button as shown above, the Properties box displays the following ID:



5. Enter a name for the object in the **Description** box. Click **OK**. VuGen inserts the new step after the selected step.

> **Note:** You can get the Control ID of the object for the purpose of pasting it into a specific location. To do this, select Copy **Control ID** from the right-click menu. You can past it into a Properties box or directly into the code from the Script view.

Add Verification Functions

When working with optional or dynamic windows or frames, you can use verification functions to determine if the window or object is available. An optional window is a window that does not consistently open during the SAP session. This function allow the Vuser script to continue running even if an optional window opens or an exception occurs.

The first example checks if a window is available. If the window is available, the Vuser closes it before continuing.

```
if (!sapgui_is_object_available("wnd[1]"))
        sapgui_call_method("{ButtonID}",
            "press",
            LAST,
            AdditionalInfo=info1011");
sapgui_press_button(.....)
```

The next example illustrates a dynamic object in the ME51N transaction. The Document overview frame is optional, and can be opened/closed by the **Document overview on/off** button.

The code checks the text on the Document overview button. If the text on the button shows Document overview on, we click the button to close the Document overview frame.

```
    if(sapgui_is_object_available("tbar[1]/btn[9]"))
    {
        sapgui_get_text("Document overview on/off button",
            "tbar[1]/btn[9]",
            "paramButtonText",
            LAST);
        if(0 == strcmp("Document overview off", lr_eval_string("
```

```
{paramButtonText}")))
            sapgui_press_button("Document overview off",
                "tbar[1]/btn[9]",
                BEGIN_OPTIONAL,
                    "AdditionalInfo=sapgui1013",
                END_OPTIONAL);
    }
```

Retrieve Information

When working with SAGUI Vusers, you can retrieve the current value of a SAP GUI object using the **sapgui_get_<xxx>** functions. You can use this value as input for another business process, or display it in the output log.

The following example illustrates how to save part of a status bar message in order to retrieve the order number.

1. Navigate to the point where you want to check the status bar text, and select **Insert New Step.** Select the **sapgui_status_bar_get_type** function. This verifies that the Vuser can successfully retrieve text from the status bar.
2. Insert an **if** statement that checks if the previous statement succeeded. If so, save the value of the argument using **sapgui_status_bar_get_param**.

   This **sapgui_status_bar_get_param** function saves the order number into a user-defined parameter. In this case, the order number is the second index of the status bar string.

```
sapgui_press_button("Save (Ctrl+S)",
    "tbar[0]/btn[11]",
    BEGIN_OPTIONAL,
        "AdditionalInfo=sapgui1038",
    END_OPTIONAL);
sapgui_status_bar_get_type("Status");
if(0==strcmp(lr_eval_string("{Status}"),"Success"))
  sapgui_status_bar_get_param("2", " Order_Number ");
```

   During test execution, the Execution log indicates the value and parameter name:

```
Action.c(240): Pressed button " Save (Ctrl+S)"
Action.c(248): The type of the status bar is "Success"
Action.c(251): The value of parameter 2 in the status bar is "33232"
```

Save Date Information

When creating scripts that use dates, your script may not run properly. For example, if you record the script on June 2, and replay it on June 3, the date fields will be incorrect. Therefore, you need to save the date to a parameter during text execution, and use the stored value as input for other date fields. To save the current date or time during script execution, use the **lr_save_datetime** function. Insert this function before the function requiring the date information. Note that the format of the date is specific to your locale. Use the relevant format within the **lr_save_datetime** function. For example, for month.day.year, specify "`%m.%d.%Y`".

In the following example, **lr_save_datetime** saves the current date. The **sapgui_set_text** function uses this value to set the delivery date for two days later.

```
lr_save_datetime("%d.%m.%Y", DATE_NOW + (2 * ONE_DAY),
             "paramDateTodayPlus2");
sapgui_set_text("Req. deliv.date",
        "{paramDateTodayPlus2}",
        "usr/ctxtRV45A-KETDAT",
        BEGIN_OPTIONAL,
            "AdditionalInfo=sapgui1025",
        END_OPTIONAL);
```

Additional SAP Resources

For more information, see the SAP website at www. sap.com or one of the following locations:

- **SAP Notes** - https://websmp103.sap-ag.de/notes

Note #480149: New profile parameter for user scripting on the front end

Note #587202: Drag =; Drop is a known limitation of the SAP GUI interface

- **SAP Patches** - https://websmp104.sap-ag.de/patches

SAP GUI for Windows - SAP GUI 6.20 Patch (the lowest allowed level is 32)

SAP GUI, SAP Web, and SAP (Click & Script) - Troubleshooting and Limitations

This section describes troubleshooting and limitations for SAP GUI, SAP Web, and SAP (Click & Script) protocols.

Troubleshooting SAP GUI Vuser Scripts

Question 1: I was able to record a script, but why does replay fail?

Answer: In LoadRunner, make sure that the LoadRunner Remote Agent is running in Process mode. Service mode is not supported. For more information, see "How to Replay SAP GUI Scripts" on page 643.

Question 2: Why were certain SAP GUI controls not recorded?

Answer: Some SAP GUI controls are only supported in their menu or toolbar contexts. Try performing the problematic task using a different means—through a menu option, context menu, toolbar, and so on.

Question 3: Why can't I record or replay any scripts in VuGen?

Answer:

1. Verify that you have the latest patch of SAP GUI 6.20 installed. The lowest allowed patch level is patch 32.
2. Make sure that scripting is enabled. See the "How to Configure the SAP Environment" on page 637.
3. Verify that notifications are disabled in the SAP GUI for Windows client. Click the Customizing of Local Layout button or press ALT+F12. Click **Options** and select the Scripting tab. Clear

both **Notify** options.

Question 4: What is the meaning of the error popup messages that are issued when I try to run the script?

Answer: Certain SAP applications store the last layout for each user (such as which frames are visible or hidden). If the stored layout was changed since the script was recorded, this may cause replay problems. For Example, in the ME52N transaction, the **Document overview Off/On** button will change the number of visible frames.

If this occurs:

1. Navigate the transaction to the same point as it was during recording, before starting replay. You can use the Snapshot viewer to see the layout in which it was recorded.
2. Add statements to the script that bring the transaction to the desired layout during replay. For example, if an optional frame interferes with your replay, insert a verification function that checks if the frame is open. If it is open, click a button to close it. For verification examples, see "How to Enhance SAP GUI Scripts" on page 645.

Question 5: Can I use the single sign-on mechanism when running a script on a remote machine?

Answer: No, VuGen does not support the single sign-on connection mechanism. In your SAP GUI client, open the Advanced Options and clear the **Enable Secure Network Communication** feature. Note that you must re-record the script after you modify the Connection preferences.

Question 6: Can VuGen record all SAP objects?

Answer: Recording is not available for objects not supported by SAP GUI Scripting. See your recording log for information about those objects.

Question 7: Are all business processes supported?

Answer: VuGen does not support business processes that invoke Microsoft Office module controls, nor those that require the use of GuiXT. You can disable **GuiXT** from the SAP GUI for Windows client Options menu.

Question 8: When I go to the Auto Logon node of the Recording Options, why is the list of server names empty?

Answer: This sometimes occurs when using SAP GUI Client 7.20. To resolve this issue, copy the **saplogon.ini** file from **%APPDATA%\SAP\Common** where **%APPDATA%** stands for the environment variable specifying the Application Data folder located directly below the user profile folder. Paste the file to the %WINDIR% folder (C:\Windows).

Siebel Web Protocol

Siebel Web Protocol Overview

The Siebel-Web protocol is similar to the standard Web Vuser protocol, with several changes in the default settings to allow it to work with the Siebel Customer Relationship Management (CRM) application.

You record typical activities in your Siebel session. VuGen records the actions and generates functions with a **web_** prefix, that emulate your actions.

Siebel Web Recording Options and Run-Time Settings

Before recording a Siebel Web Vuser script, set the following recording options:

- **Recording** node: **HTML-based script**

  HTML Advanced - Script type: **A script containing explicit URLs only**

  HTML Advanced - Non HTML-generated elements: **Do not record**

- **Advanced** node: Clear the **Reset context for each action** check box.

  Before running a Siebel Web Vuser script, set the following run-time setting:

  In the Run-Time settings, clear the **Simulate a new user on each iteration** check box in the **Browser Emulation** node.

How to Record Transaction Breakdown Information

VuGen provides a diagnostic tool for understanding the transaction components in your test—**transaction breakdown**. Using transaction breakdown, you can determine where the bottlenecks are and the issues that need to be resolved.

When preparing your script for transaction breakdown, we recommend that you add think time steps at the end of each transaction using the ratio of one second per hour of testing. For more information about adding think time steps, see "How to Insert Steps into a Script" on page 207.

In order to record the transaction breakdown information, you need to modify your the parameterization functions in your script.

Prepare Your Script for Transaction Breakdown

1. Identify the script parameterization replacement of the Session ID.

```
/* Registering parameter(s) from source task id 15
   // {Siebel_sn_body4} = "28eMu9uzkn.YGFFevN1FdrCfCCOc8c_"
   // */
   web_reg_save_param("Siebel_sn_body4",
       "LB/IC=_sn=",
       "RB/IC==;",
       "Ord=1",
       "Search=Body",
       "RelFrameId=1",
       LAST);
```

2. Mark the next **web_submit_data** function as a transaction by enclosing it with **lr_start_transaction** and **lr_end_transaction** functions.

3. Before the end of the transactions, add a call to **lr_transaction_instance_add_info**, where the first parameter, 0 is mandatory and the session ID has a SSQLBD prefix.

```
lr_start_transaction("LoginSQLSync");
    web_submit_data("start.swe_2",
```

```
            "Action=http://design/callcenter_enu/start.swe",
            "Method=POST",
            "RecContentType=text/html",
            "Referer=http://design/callcenter_enu/start.swe",
            "Snapshot=t2.inf",
            "Mode=HTML",
            ITEMDATA,
            "Name=SWEUserName", "Value=wrun", ENDITEM,
            "Name=SWEPassword", "Value=wrun", ENDITEM,
            "Name=SWERememberUser", "Value=Yes", ENDITEM,
            "Name=SWENeedContext", "Value=false", ENDITEM,
            "Name=SWEFo", "Value=SWEEntryForm", ENDITEM,
            "Name=SWETS", "Value={SiebelTimeStamp}", ENDITEM,
            "Name=SWECmd", "Value=ExecuteLogin", ENDITEM,
            "Name=SWEBID", "Value=-1", ENDITEM,
            "Name=SWEC", "Value=0", ENDITEM,
            LAST);
lr_transaction_instance_add_info(0,lr_eval_string("SSQLBD:{Siebel_
sn_body4}"));
lr_end_transaction("LoginSQLSync", LR_AUTO);
```

Note: To avoid session ID conflicts, make sure that the Vusers log off from the database at the end of each session.

Siebel Web - Troubleshooting and Limitations

This section describes troubleshooting and limitations for Siebel Web Vuser scripts.

Back or Refresh Error

An error message relating to **Back or Refresh** typically has the following text:

We are unable to process your request. This is most likely because you used the browser back or refresh button to get to this point.

Cause: The possible causes of this problem may be:

- The SWEC was not correlated correctly for the current request.
- The SWETS was not correlated correctly for the current request.
- The request was submitted twice to the Siebel server without the SWEC being updated.
- A previous request should have opened a frame for the browser to download. This frame was not created on the server probably because the SWEMethod has changed since the recording.

Same Values

A typical Web page response to the **Same Values** error is:

@0`0`3`3``0`UC`1`Status`Error`SWEC`10`0`1`Errors`0`2`0`Level0`0`ErrMsg`The same values for 'Name' already exist. If you would like to enter a new record, please make sure that the field values are unique.`ErrCode`28591`

Cause: The possible cause of this problem may be that one of the values in the request (in the above example, the value of the Name field) duplicates a value in another row of the database table. This value needs to be replaced with a unique value to be used for each iteration per user. The recommended solution is to replace the row ID with its parameter instead insuring that it will be unique.

No Content HTTP Response

A typical HTTP response for a **No Content HTTP Response** type error is:

HTTP/1.1 204 No Content

Server: Microsoft-IIS/5.0

Date: Fri, 31 Jan 2003 21:52:30 GMT

Content-Language: en

Cache-Control: no-cache

Cause: The possible causes of this problem may be that the row ID is not correlated at all or that it is correlated incorrectly.

Restoring the Context

The typical Web page response to the **Restoring the Context** type error is:

@0`0`3`3``0`UC`1`Status`Error`SWEC`9`0`1`Errors`0`2`0`Level0`0`ErrMsg`An error happened during restoring the context for requested location`ErrCode`27631`

Cause: The possible causes of this problem may be that the rowid is not correlated or that it is correlated incorrectly.

Cannot Locate Record

The typical Web page response to the **Cannot locate record** type error is:

@0`0`3`3``0`UC`1`Status`Error`SWEC`23`0`2`Errors`0`2`0`Level0`0`ErrMsg`Cannot locate record within view: Contact Detail - Opportunities View applet: Opportunity List Applet.`ErrCode`27573`

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

End of File

The typical Web page response to the **End of File** type error is:

@0`0`3`3``0`UC`1`Status`Error`SWEC`28`0`1`Errors`0`2`0`Level0`0`ErrMsg`An end of file error has occurred. Please continue or ask your systems administrator to check your application configuration if the problem persists.`ErrCode`28601`

Cause: The possible causes of this problem may be that the input name SWERowId does not contain a row ID for a record on the Web page. This input name should have been parameterized. The parameter's source value may have changed its location.

Unable to Retrieve Search Categories

The typical Web page response to the **Unable to Retrieve Search Categories** type error is:

Cause: A possible cause of this problem may be that the search frame was not downloaded from the server. This occurs when the previous request did not ask the server to create the search frame correctly.

SilverLight Protocol

Silverlight Protocol - Overview

Microsoft Silverlight is a web application framework that supports graphics, animations, and interactivity. The Silverlight protocol enables you to record applications built with Microsoft Silverlight. The Silverlight protocol includes the Web (HTTP/HTML) protocol as a subset, as well as a number of additional functions, recording options, and run-time settings.

In order to record high level Vuser scripts, you can import WSDL files used by your application in the recording options.

How to Import WSDL Files

The following steps describe how to import WSDL files into a Silverlight Vuser script, manually or automatically. Alternatively, you can disable WSDL files and generate soap requests. All of these options are performed in the **Silverlight > Services** node of the **Recording Options Dialog Box**. For user interface details, see "Silverlight > Services Node" on page 327.

Automatically Locate WSDL Files

To configure VuGen to automatically detect the WSDL files used by your script and attempt to locate them, select **Use WSDL files included in the script** and **Automatically detect WSDL files and import services during code generation.** If a WSDL is detected that cannot be imported, you will be notified in the Code Generation Notifications box. Ofra: is that the right name for the dialog box?

Manually Locate WSDL Files

You can manually locate WSDL files in a number of ways from the Add Service Dialog Box. To locate a WSDL file whose URL is known, use the **URL** option. If the WSDL file is on your local machine, you the **File** option. To search for the WSDL in the WSDL History (a list of previously imported WSDLs), select **Previously Imported** and click **...** to open the list.

For user interface details, see "Add / Edit Services Dialog Box" on page 328.

Disable WSDL Files

You can disable WSDL files and generate SOAP requests instead. This results in a lower level script, however it does increase the performance of your script. To disable WSDL files, select **Do not use WSDL files**.

Advanced Security Settings

You can modify security and password settings in the Protocol and Security Scenario Data dialog box. For details, see "Protocol and Security Scenario Data Dialog Box" on page 329.

Silverlight - Troubleshooting and Limitations

While recording a site developed in Silverlight, the Install Silverlight step is recorded even though recording process did not include installing the Silverlight plug-in.

Workaround

Configure the run-time settings to exclude the following address:
http://go.microsoft.com/fwlink/?LinkId=108181

1. Select the **Run-Time Settings > Internet Protocol > Download Filters** Node.
2. Select the **Exclude addresses in list** radio button.
3. Click **Add** and add http://go.microsoft.com/fwlink/?LinkId=108181 to the list.

Note: The ?LinkId=108181 portion of the URL address may not be static over time and may need to be updated.

Tuxedo Protocols

Tuxedo Protocol - Overview

When you record a Tuxedo application, VuGen generates LRT functions that describe the recorded actions. These functions emulate communication between a Tuxedo client and a server. Each Tuxedo function begins with an **lrt,tp, tx,** or **F** prefix**.**

In the following example, VuGen recorded a client's actions in a Tuxedo bank application. The client performed an action of opening a bank account and specifying all the necessary details. The session was aborted when the client specified a zero opening balance.

```
 lrt_abort_on_error();
lr_think_time(65);
tpresult_int = lrt_tpbegin(30, 0);
data_0 = lrt_tpalloc("FML", "", 512);
lrt_Finitialize((FBFR*)data_0);
/* Fill the data buffer data_0 with new account information */
lrt_Fadd_fld((FBFR*)data_0, "name=BRANCH_ID", "value=8", LRT_END_OF_
PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ACCT_TYPE", "value=C", LRT_END_OF_
PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=MID_INIT", "value=Q", LRT_END_OF_
PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=PHONE","value=123-456-7890",
                   LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=ADDRESS", "value=1 Broadway
                   New York, NY 10000", LRT_END_OF_PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=SSN","value=111111111", LRT_END_OF_
PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=LAST_NAME", "value=Doe",LRT_END_OF_
```

```
PARMS);
lrt_Fadd_fld((FBFR*)data_0,"name=FIRST_NAME", "value=BJ",LRT_END_OF_
PARMS);
lrt_Fadd_fld((FBFR*)data_0, "name=SAMOUNT", "value=0.00",LRT_END_OF_
PARMS);
/* Open a new account */
tpresult_int = lrt_tpcall("OPEN_ACCT", data_0, 0, =;data_0, =;olen_2,
0);
lrt_tpabort(0);
lrt_tpcommit(0);
lrt_tpfree(data_0);
lrt_tpterm();
```

Notes About Working with Tuxedo Scripts

The following important notes should be reviewed before recording and running a tuxedo script:

- We recommend using the Tuxedo 6 protocol for recording Tuxedo 6.x and earlier, and the Tuxedo protocol for Tuxedo 7.x and higher.
- Before you record, verify that the Tuxedo folder, %TUXDIR%\bin is in the path.
- If the environment variables have changed since the last time you restarted VuGen, VuGen may record the original variable value rather than the current value.
- To avoid any inconsistencies, you should restart VuGen before recording Tuxedo applications.
- To run PeopleSoft-Tuxedo Vusers with Tuxedo 7.x, you must change the library extension in the *mdrv.dat* file as follows:

```
[PeopleSoft-Tuxedo]
WINNT_EXT_LIBS=lrt7.dll
```

Defining Environment Settings for Tuxedo Vusers

The following section describes the system variable settings for Tuxedo Vusers running on Windows and Linux platforms. You define the system variables in your Control Panel/System dialog box (NT) or .cshrc or .login file (Linux).

TUXDIR	the root folder for Tuxedo sources.
FLDTBLDIR	list of directories containing FML buffer information. In Windows, separate the names of directories with semi-colons. On Linux platforms, separate the names of the directories with a colon.
FIELDTBLS	list of files containing FML buffer information. On both Windows and Linux platforms, separate the file names with commas.

For example:

```
SET FLDTBLDIR=%TUXDIR%\udataobj;%TUXDIR%\APPS\WS (PC)
SET FIELDTBLS=bankflds,usysflds (PC)
```

```
setenv FLDTBLDIR $TUXDIR/udataobj:$TUXDIR/apps/bankapp (Linux)
setenv FIELDTBLS bank.flds,Usysflds (Linux)
```

You must define the following system variables for Tuxedo clients using Tuxedo/WS workstation extensions during execution:

WSNADDR	specifies the network address of the workstation listener process. This enables the client application to access Tuxedo. Note that to define multiple addresses in a WSNADDR statement, each address must be separated by a comma.
WSDEVICE	specifies the device that accesses the network. Note that you do not need to define this variable for some network protocols.

For example:

```
SET WSNADDR=0x0002ffffc7cb4e4a (PC)
setenv WSNADDR 0x0002ffffc7cb4e4a (Linux)
setenv WSDEVICE /dev/tcp (Linux)
```

Tuxedo Buffer Data

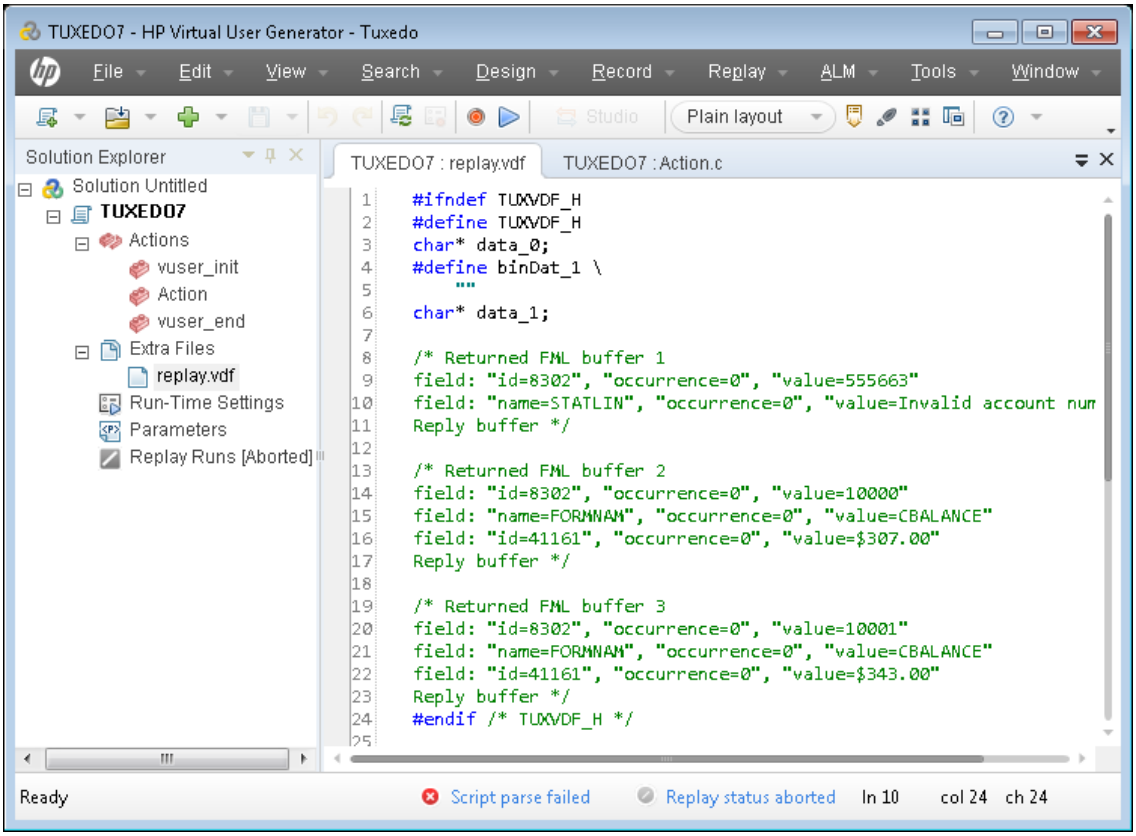
When you use VuGen to create a Tuxedo Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Actions**, and **vuser_end**.

The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file is called **replay** .**vdf** , and it contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRT functions use the buffer descriptors to access the data.

You can use VuGen to view the contents of the data file by selecting the **replay.vdf** file in the **Solution Explorer**.

The option to view a data file is available by default for Tuxedo scripts.



Tuxedo and Tuxedo 6 - Troubleshooting and Limitations

This section describes troubleshooting and limitations for Tuxedo and Tuxedo 6 Vusers.

- If you encounter problems recording or replaying Tuxedo applications, or the script is missing a call to lrt_tpinitialize, contact Customer Support to check which DLLs are used with the application.
- If the application uses **wtuxws32.dll**, instead of **libwsc.dll**, contact Customer Support to obtain a patch to enable the recording.
- If you encounter problems recording or running Tuxedo applications, check that the Tuxedo application runs without VuGen, and that the environment variables have been defined correctly. For more information, see "Tuxedo Buffer Data" on previous page. Note that after you set or modify the Tuxedo variables, you should restart VuGen and your application, in order for the changes to take effect. If your application is 16-bit, then you also need to kill the NTVDM process.
- If you experience problems during execution, check the Tuxedo log file on the side of the server for error messages. By default, this file is found in the folder indicated by the environment variable APPDIR. The file name has the form ULOG.mmddyy, where mmddyy indicates the current month, day, and year. The file for March 12, 1999 would be ULOG.031299. The default location of this file can be changed by setting the environment variable ULOGPFX on the server. A log file can also be found on the client side, in the current folder, unless the ULOGPFX variable changes its location.

Web (Click & Script) Protocol

Web (Click & Script) Enhancements

This topic includes several enhancements that can assist you in creating your Vuser scripts.

Most of the features described below are enhancements to the API functions. For detailed information about the functions and their arguments, see the Function Reference (**Help > Function Reference**), or click F1 on any function in the Editor.

Adding Conditional Steps

The Web (Click & Script) functions, **web_xxxx**, allow you to specify conditional actions during replay. Conditions are useful, for example, if you need to check for an element and perform an action only if the element is found.

For example, suppose you perform an Internet search and you want to navigate to all of the result pages by clicking Next. Since you do not know how many result pages there will be, you need to check if there is a Next button, indicating another page, without failing the step. The following code adds a verification step with a notification—if it finds the Next button, it clicks on it.

```
While (web_text_link("Next",
DESCRIPTION,
    "Text=Next",
    VERIFICATION,
    "NotFound=Notify",
    ACTION,     "UserAction=Click",
    LAST) == LR_PASS);
```

For details about the syntax and use of the VERIFICATION section, see the Function Reference (**Help > Function Reference**).

Checking a Page Title

In **web_browser** steps, you can use the title verification recording option to make sure that the correct page is downloaded. You can instruct the Vuser to perform this check automatically for every step or every navigation to a new top level window.

In addition, you can manually add title verifications to your script at the desired locations, using both exact and regular expression matches.

```
web_browser("test_step",
DESCRIPTION,
...
VERIFICATION,
   "BrowserTitle=Title",
   ACTION,]
,
LAST);
```

For more information, see theFunction Reference (**Help > Function Reference**).

You can set title verification options directly from within the Recording options. For more information, see the section about recording with Click & Script.

Text Check Verification

Using text checkpoints, you can verify that a text string is displayed in the appropriate place on a Web page or application and then perform an action based on the findings. You can check that a text string exists (**ContainsText**),or that it does not exist (**DoesNotContainText**),using exact or regular expression matching.

For example, suppose a Web page displays the sentence "Flight departing from New York to San Francisco". You can create a text checkpoint that checks that the words "New York" are displayed between "Flight departing from" and "to San Francisco". (In this example, you would need to use regular expression criteria.)

To implement these checkpoints, you add the Text Check related arguments to the VERIFICATION section of the step. During replay, Vusers search the innerText of the browser's HTML document and any child frames. The **NotFound** argument specifies the action to take if verification fails, either because the object was not found or because the text verification failed: Error, Warning, or Notify.

You can manually add text verifications to your script for existing steps. Place the text verification after the step that generated the element.

The text validation arguments are valid for the following Action functions: **web_browser**, **web_element**, **web_list**, **web_text_link**, **web_table**, and **web_text_area**.

Note: You can use the same type of text verification only once per step (for example, **ContainsText** twice). If you want to check for multiple texts, separate them into several steps. You can, however, use different verifications in the same step (for example, **ContainsText** =; **DoesNotContainText**). In this case, all conditions have to be met in order for the step to pass.

In the following example, the verification arguments check that we were not directed from www.acme.com to the French version of the website, acme.com/fr.

```
web_browser("www.acme.com",
        ACTION,
        "Navigate=http://www.acme.com/",
    LAST);
web_browser("Verify",
        VERIFICATION,
        "ContainsText=Go to Acme France",
        "DoesNotContainText=acme.com in English",
    LAST);
```

Saving a Java Script Value to a Parameter

The **EvalJavaScript** argument lets you evaluate Java Script on the Web page.

Suppose you want to click on a link which has the same name as the page title. The following example evaluates the document title and uses it in the next **web_text_link** function.

```
web_browser("GetTitle",
    ACTION,
    "EvalJavaScript=document.title;",
    "EvalJavaScriptResultParam=title",
    LAST);
```

```
web_text_link("Link",
    DESCRIPTION,
    "Text={title}",
    LAST);
```

Working with Custom Descriptions

Suppose you want to randomly click a link that belongs to some group. For example, on **hp.com** you want to randomly select a country. Regular description matching will not allow this type of operation. However, using a custom description argument, you can identify the group with an attribute that is common to all the links in the group.

Using the custom description argument, you specify any attribute of the element, even those that are not predefined for that element. During replay, the Vuser searches for those attributes specified in the DESCRIPTION section. Replay will not fail on any unknown argument in the DESCRIPTION section.

For example, to find the following hyperlink:

Yahoo, use:

```
web_text_link("yahoo",
    DESCRIPTION,
    "Text=yahoo",
    "my_attribute=bar",
    LAST);
```

In the following example, since all the relevant links have the same class name, newmerc-left-ct, you can perform a random click using the following code:

```
web_text_link("Click",
                        DESCRIPTION,
                        "Class=newmerc-left-ct",
                        "Ordinal=random",
                        LAST);
```

The following functions do not support the custom description arguments: **web_browser**, **web_map_area**, **web_radio_group**, and **web_reg_dialog**.

Copying Text to the Clipboard

When you work with Web (Click & Script) Vuser scripts, VuGen lets you copy text from a snapshot to the clipboard. This functionality is available in both the Page view and Page Source view. For details on how to copy the text to the clipboard, see "How to Work with Snapshots" on page 57.

Web (Click & Script) Example Script

Click & Script Vuser scripts typically contain several actions which make up a business process. By viewing the recorded functions that were generated on a GUI level, you can determine the user's exact actions during the recorded session.

For example, in a typical recording, the first stage may contain a sign-in process. The browser opens on the sign-in page, and a user signs in by submitting a user name and password and clicking **Sign In**.

For Web (Click & Script) Vuser scripts, VuGen generates a **web_edit_field** function that represents the data entered into an edit field. In the example that follows, a user entered text into the userid field, and a password into the pwd field which is encrypted.

```
vuser_init() {
    web_browser("WebTours",
        DESCRIPTION,
        ACTION,
        "Navigate=http://localhost:1080/WebTours/",
        LAST);
    web_edit_field("username",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=text",
        "Name=username",
        "FrameName=navbar",
        ACTION,
        "SetValue=jojo",
        LAST);
    web_edit_field("password",
        "Snapshot=t3.inf",
        DESCRIPTION,
        "Type=password",
        "Name=password",
        "FrameName=navbar",
        ACTION,
        "SetEncryptedValue=440315c7c093c20e",
        LAST);...
```

Web (Click & Script) API Notes

This section lists general notes about the Web functions. Note that you can specify a regular expression for most object descriptions, by preceding the text with "/RE" before the equals sign. See the Function Reference (**Help > Function Reference**) for more details. For example:

```
web_text_link("Manage Assets",
        DESCRIPTION,
        "Text/RE=(Manage Assets)|(Configure Assets)",
        ACTION,
        "UserAction=Click",
        LAST);
```

Ordinals

The Ordinal attribute is a one-based index to distinguish between multiple occurrences of objects with identical descriptions. In the following example, the two recorded **web_text_link** functions have identical arguments, except for the ordinal. The ordinal value of 2, indicates the second occurrence.

```
web_text_link("Manage Assets",
        DESCRIPTION,
```

```
        "Text=Manage Assets",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Ordinal=2",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

Empty Strings

There is a difference between not specifying an argument and specifying it as an empty string. When you do not specify an argument, VuGen uses the default value or ignores it. When you list an argument, but assign it an empty string as a value, VuGen attempts to find a match with an empty string or no string at all. For example, in a **web_text_link** function, omitting the id argument instructs VuGen to ignore the id property of the HTML element. Specifying "ID=" searches for HTML elements with no id property or with an empty ID.

```
web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Id=",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

Click & Script - Recording Tips

This section lists tips for recording click-and-script Vuser scripts.

Use the Mouse and not the Keyboard

It is preferable to click on an object with the mouse rather than using the keyboard. During recording, use only GUI objects that are within the browser's pane. Do not use any browser icons, controls, the Stop button, or menu items, such as **View > Refresh**. You may, however, use the Refresh, Home, Back and Forward buttons and the address bar.

Do not Record Over an Existing Script

It is best to record into a newly created script—not an existing one.

Avoid Context Menus

Avoid using context menus during recording. Context menus are right-click menus which pop up when clicking certain objects in a graphical user interface.

Avoid Working in Another Browser While Recording

While recording, do not work in any browser window other than the browser windows opened by VuGen.

Wait for Downloads

Wait for all downloads to complete before doing any action, such as clicking on a button or filling in a text field.

Wait for Pages to Load

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Navigate to the Start Page

If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page at the end of your recording, so that the same link will be visible at the end of the business process.

Use a Higher Event Configuration Level

Record the business process again using the **High** event configuration level. For more information on changing the event configuration level, see "Click and Script - Troubleshooting and Limitations" on page 666.

Disable Socket Level Recording

In certain cases, the capturing of the socket level messages disrupts the application. For most recordings, socket level data is not required. To prevent the recording of socket level data, disable the option in the recording options. For more information, see "GUI Properties > Advanced Node" on page 307.

Enable the "Record rendering-related property values" Option

If the client-side scripts of the application use a lot of styling activities, enable the **Record rendering-related property values** option before recording the script. For example, enable this option to record additional DOM properties such as **offsetTop**. Note that enabling this option may decrease the recording speed. You can enable the option by selecting **Recording Options > GUI Properties > Advanced**. For more information, see "GUI Properties > Advanced Node" on page 307.

Click & Script - Replay Tips

This section lists tips for replaying click-and-script Vuser scripts.

Do not Reorder Statements in a Recorded Script

Do not change the order of the statements within a recorded script. Also, copying segments of code from one Action to another is not recommended.

Convert non-ASCII Characters

If your links contain non-ASCII characters, you should instruct VuGen to convert the data to or from the UTF-8 format.

Enable UTF-8 Conversion

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate the **Convert from/to UTF-8** option and set it to **Yes**.

Alternatively, view the list of options that is displayed when a link is not found. Enter the displayed text as-is, such as the hex escape sequences \xA0 or any other non-standard format.

Run the Same Sequence of Actions Twice

In some cases, you can perform a certain process only once—such as deleting a user from the database. Replay will fail after the first iteration because the action is no longer valid. Verify that your business process can be repeated more than once with the same data.

Set Unique Image Properties

In the Step Navigator, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you can add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the Function Reference (**Help > Function Reference**).

Check the Step's Description

If you receive a **GUI Object is not found** error, check the Output pane for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the script in the Editor and manually modify the value of the step's DESCRIPTION argument.
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument. For more information, see the Function Reference (**Help > Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property. For more information, see the Function Reference (**Help > Function Reference**).

Click & Script - Miscellaneous Tips

The following additional tips may help you in troubleshooting problems that you experience with click-and-script Vuser scripts:

Search for Warnings

Search for warnings or alerts in the Output pane.

Verify the Response

Verify the response of the previous step is correct using **web_reg_find**. For more information, see the Function Reference (**Help > Function Reference)**.

Use Alternate Navigation

For problematic steps or those using Java applets, use **Alternative Navigation** to replace the Web step with an HTTP level step. Note that the HTTP level steps may require manual correlations. To perform Alternative Navigation, select a step in the **Step Navigator**, or the text in Script View, and select **Replace with alternative navigation** from the right-click menu.

Working with the Kerberos Protocol

If you are using the Kerberos Protocol for authentication, you must customize VuGen to properly convene authorization sessions. Advanced users can attempt to perform this customization themselves.

In order for the Kerberos Protocol to work properly, create a krb5.ini file and put it in an available folder. Save the full path name of krb5.ini into the KRB5_CONFIG environment variable.

The krb5.ini file should contain detailed information about each domain (KDS and AS addresses) and trust chains.

For more information, contact HP software support.

Click and Script - Troubleshooting and Limitations

This section describes troubleshooting and limitations for click-and-script protocols. Some items apply to specific click-and-script protocols only.

Recording Issues and Limitations

Firefox is not supported

Only Internet Explorer is supported for Web (Click & Script). To record browser activity on Firefox, use the Web (HTTP/HTML) protocol.

Application behaves differently while being recorded

If your application behaves differently during recording, than it does without recording, you should determine if the recording problem is unique to Web. The effect may be that a Web page will not load, part of the content is missing, a popup window does not open, and so forth.

Create a new Web (HTTP/HTML) script and repeat the recording.

In the event that the recording fails in Web (HTTP/HTML), we recommend that you disable socket level recording (see "Click & Script - Recording Tips" on page 663).

The problem may be the result of an event listener. Use trial and error to disable event listeners in the **Web Event Configuration** Recording Options, and then re-record your session as a Web user.

Certain Click and Script steps do not generate properly

After recording a script, if not all steps are correctly generated, the problem may be due to the **Windows Component > Internet Explorer Enhanced Security Configuration**.

Remove **Internet Explorer Enhanced Security Configuration** by selecting **Control Panel > Add or Remove Programs > Add or Remove Windows Components** and re-record your script.

Disable an Event Listener

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Recording > Recording Options** and select the **GUI Properties > Web Event Configuration** node.
3. Click **Custom Settings** and expand the **Web Objects** node. Select an object.
4. Select **Disabled** from the list in the **Record** column for the relevant Web object. If the recording still does not work, enable the listener you previously disabled, and try disabling another one. Repeat these steps until your recording succeeds.

Dynamic menu navigation was not recorded

A dynamic menu is a menu that dynamically changes depending on where you select it. If the dynamic menu navigation was not recorded, record again using "high" event configuration mode. These settings can be found in the **Recording Options > GUI Properties > Web Event Configuration** node.

Certain user actions were not recorded

Check if there is a Java applet running inside the browser. If not, record the script with the Web (HTTP/HTML) protocol.

Replay Issues

GUI object not found

Does the error occur at the beginning of the second iteration?

If the error occurs at the beginning of the second iteration's Action section, it is probably the result of a starting page that was present for the first iteration, but missing for the second one. If the last page in an action does not contain the links and buttons that were available at the start of the iteration, then the next iteration will fail. For example, if the first page has a text link **Book A Flight**, make sure to navigate to the appropriate page, so that the same link will be visible at the end of the business process.

Is it a text link containing non-ASCII characters?

If the problem occurs with non-ASCII characters, you should instruct VuGen to covert the data to a suitable character set.

Enable Data Conversion on Windows Machines

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate **Charset Conversions by HTTP** in the Web (Click & Script) > General options, and set it to **Yes**.

Enable UTF-8 conversion for Linux Machines

Click **Record > Recording Options** to open the Recording Options dialog box.

Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.

Click **Options** to open the Advanced Options dialog box.

Locate **Convert from/to UTF-8** in the General options and set it to **Yes**.

Alternatively, view the list of alternatives that are displayed when a link is not found. Enter the displayed text as-is, such as hex escape sequences \xA0 or any other non-standard format.

Can you run the same sequence of actions twice in the application?

In some cases, you can only perform a certain process once, such as deleting a user from the database. Replay will fail after the first iteration, because the action is no longer valid. Verify that your business process can be repeated in the application more than once with the same data, without recording again.

Were the image properties 'Id', 'Name' and 'Alt' empty?

In the **Step Navigator**, double click on the previous image step to open its properties. If the **Id**, **Name**, and **Alt** properties are empty, provide further identification of the image, such as its file name in the **Src** property.

Alternatively, you add an **Ordinal** argument to specify the occurrence number of the image on that page. The **Ordinal** argument uniquely identifies each image on the page where all other identification arguments are not unique. For more information, see the Function Reference (**Help > Function Reference**).

Did the step's description change?

Check the Output pane for a list of the objects in the problematic step. In some cases, the object description changes slightly from run to run.

There are several solutions:

- If the new value is stable, open the Script View and manually modify the value of the step's DESCRIPTION argument(s).
- If the description changes from run to run, you can use a regular expression in the DESCRIPTION argument(s). For more information, see the Function Reference (**Help > Function Reference**).
- Alternatively, replace the problematic object description property, such as Name, with the Ordinal property For more information, see the Function Reference (**Help > Function Reference**).

Did the page load completely during recording?

During recording, it is best to wait for the page to load completely before doing the next step. If you did not wait for all of the pages to load, record the script again.

Replay failure

If the replay is failing at a particular step, check the step description. VuGen sometimes reads a single space as a double space. Make sure that there are no incorrect double spaces in the string.

Miscellaneous Issues

Out of memory error in JavaScript

Increase the JavaScript memory in the run-time settings.

Increase the JavaScript Memory Size

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Replay > Run-Time Settings** and select the **Internet Protocol > Preferences** node.
3. Click **Options** to open the Advanced Options dialog box.
4. Locate the **Memory Management JavaScript Runtime Memory Size (Kb)** and **Memory Management JavaScript Stack Memory Size (Kb)** options.
5. Increase the memory sizes to 512Kb or higher.

VuGen displays JavaScript errors

If VuGen displays JavaScript errors in the Output pane, enable IE (Internet Explorer) script errors in order to verify that the Javascript itself does not contain errors.

Show Script Errors

1. Open Internet Explorer.
2. Select **Tools > Internet Options** and click the **Advanced** tab.
3. Under **Browsing**, select the **Display a notification about every script error** check box.
4. Rerun the application in IE. If IE displays script errors, then there is a problem with the JavaScript application. If it is not possible to fix the application, you can safely ignore the corresponding replay errors.

Problems following parameterization

If you encounter problems only after you have parameterized values, verify that the values are valid for your application. Perform business process with the value of the parameter and verify that the application accepts it.

Problems with applications that utilize styling actions

If the client-side scripts of the application use a lot of styling activities, you should record the script again after enabling the **Record rendering-related property values** option. This enables the recording of additional DOM objects.

Enable the "Record rendering-related property values" Option

1. Click **Record > Recording Options** to open the Recording Options dialog box.
2. Select **Recording > Recording Options** and select the **GUI Properties > Advanced** node.
3. Select the **Record rendering-related property values** check box.

Re-record the Vuser script.

Web Protocols

Web Protocols Overview

You use VuGen to develop Web Vuser scripts. While you navigate through a site performing typical user activities, VuGen records your actions and generates a Vuser script. When you run the script, the resulting Vuser emulates a user accessing the Internet.

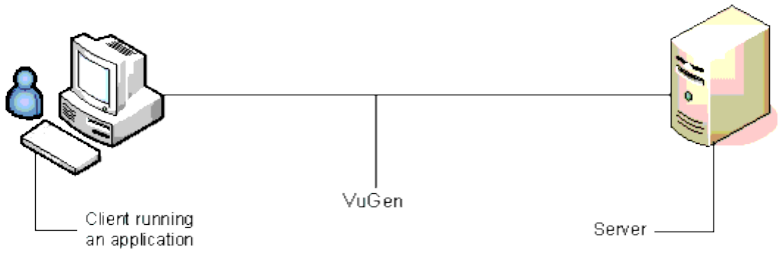
Suppose you have a web site that displays product information for your company. The site is accessed by customers and potential customers. You want to make sure that the response time for any customer query is less than a specified value (for example, 20 seconds)—even when many users (for example, 200) access the site simultaneously. You use Vusers to emulate this scenario, where the web server services simultaneous requests for information. Each Vuser could do the following:

- Load the home page
- Navigate to the page containing the product information
- Submit a query
- Wait for a response from the server

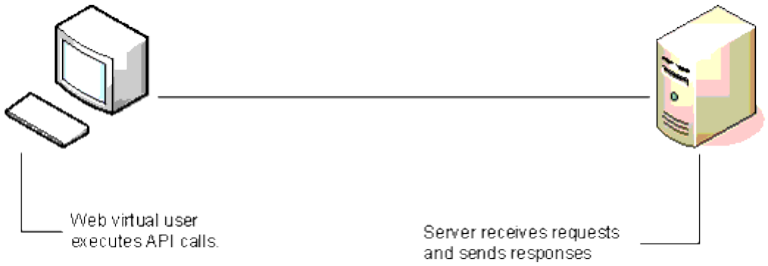
You can distribute several hundred Vusers among the available testing machines, each Vuser accessing the server by using its API. This enables you to measure the performance of the server under the load of many Vusers.

Web Vuser Technology

VuGen creates Web Vuser scripts by recording the activity between a browser and a Web server. VuGen monitors the client (browser) end of the system and traces all the requests sent to, and received from, the server.



When you run a recorded Vuser script, the Vuser communicates directly with the server without relying on client software. Instead, the Vuser script executes calls directly to the Web server via API functions.



Web Vuser Types

To create a Web Vuser script, select one of the following Web Vuser protocols:

Web (Click & Script)

The Web (Click & Script) protocol records Web sessions on a user-action GUI level. VuGen creates a GUI-level script that intuitively represents actions in the Web interface. For example, VuGen generates a **web_button** function when you click a button to submit information, and VuGen generates a **web_edit_field** function when you enter text into an edit box.

Web (Click & Script) Vusers support non-HTML code such as Javascript on the client side. VuGen creates an intuitive script that emulates your actions on the web page, and executes the necessary Javascript code.

Web (Click & Script) Vusers handle most correlations automatically, reducing the time required to create the script. In most cases, you do not need to define rules for correlations or perform manual correlations after the recording.

Web (Click & Script) Vusers allow you to generate detailed Business Process Reports which summarize the script. For example, when you click a button to submit data, VuGen generates a **web_button** function. If the button is an image, VuGen generates a **web_image_submit** function. In the following example, a Vuser clicks the **Login** button.

```
...
web_image_submit("Login",
        "Snapshot=t4.inf",
        DESCRIPTION,
        "Alt=Login",
        "Name=login",
        "FrameName=navbar",
        ACTION,
        "ClickCoordinates=31,6",
        LAST);}
```

The next section illustrates a user navigating to the Asset ExpressAdd process under the Manage Assets branch. The user navigates by clicking the text links of the desired branches, generating **web_text_link** functions.

```
web_text_link("Manage Assets_2",
        DESCRIPTION,
        "Text=Manage Assets",
        "Ordinal=2",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
web_text_link("Use",
        DESCRIPTION,
        "Text=Use",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
```

```
        LAST);
web_text_link("Asset ExpressAdd",
        DESCRIPTION,
        "Text=Asset ExpressAdd",
        "FrameName=main",
        ACTION,
        "UserAction=Click",
        LAST);
```

In the following example, the **web_list** function emulates the selection of a list item.

```
...
web_list("Year",
        DESCRIPTION,
        "Name=Year",
        "FrameName=CalFrame",
        ACTION,
        "Select=2000",
        LAST);
```

When you click on an image that is associated with an image map, VuGen generates a **web_map_area** function.

```
web_map_area("map2_2",
        DESCRIPTION,
        "MapName=map2",
        "Ordinal=20",
        "FrameName=CalFrame",
        ACTION,
        "UserAction=Click",
        LAST);
```

Note: Web (Click & Script) Vusers do not support applets or VB script. If the web site that is accessed by the Vusers contains these items, use the Web (HTTP/HTML) protocol.

For more information about Web (Click & Script) Vuser scripts, see Click & Script Protocols.

Web (HTTP/HTML)

When recording a Web (HTTP/HTML) Vuser script, VuGen records the HTTP traffic between the browser and the server. The scripts contain detailed information about the recorded traffic.

The Web (HTTP/HTML) Vuser protocol provides two recording levels: **HTML-based script** and **URL-based script**. These levels determine what information to record and which functions to use when generating the Vuser script. For more information about selecting a recording level, see "Recording Levels Overview" on page 261.

Tip: For most applications, including those with JavaScript, use the Web (Click & Script) Vuser protocol. For browser applications that include applets and VB script or for non-browser applications, use the Web (HTTP/HTML) Vuser protocol.

Working with Cache Data

You can save stored data into your browser's cache, and load it at a later point in the script.

To implement this within your script, you manually add the **web_dump_cache** and **web_load_cache** functions.

For task details, see "How to Insert Caching Functions" on page 679.

Dumping Information to the Cache

Transferring data to the cache is called dumping the information. You run the **web_dump_cache** function to create a cache file in the location specified in the **FileName** argument. You need to run this function only once to generate the cache file.

In the following example, the **web_dump_cache** function creates a cache file in C:\temp for each VuserName parameter running the script.

```
web_dump_cache("paycheckcache","FileName=c:\\temp\\{Vuser
        Name}paycheck", "Replace=yes", LAST)
```

If you run a single Vuser user ten times, VuGen creates ten cache files in the following format, where the prefix is the VuserName value:

```
Ku001paycheck.cache
Ku002paycheck.cache
Ku003paycheck.cache
...
```

You can modify the first and second arguments (`paycheckcache` and `paycheck` in this example) to reflect the current transaction name. Place this function at the end of your script, after you have loaded all of the resources.

Loading Information from the Cache

The **web_load_cache** function loads a cache file whose location is specified in the **FileName** argument. Note that the **web_load_cache** function requires the cache file to exist. Therefore, you can run this function only after running **web_dump_cache**.

In the following example, the **web_load_cache** function loads the **paycheck** cache files from **C:\temp**.

```
web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",
LAST)
```

Text and Image Verification

VuGen enables you to add checks to your Web Vuser scripts. A check verifies the presence of a specific object on a Web page. The object can be a text string or an image.

Web checks enable you to determine whether or not your Web site is functioning correctly while it is being accessed by many Vusers—that is, does the server return the correct Web pages? This is particularly important while your site is under the load of many users, a period when the server is more likely to return incorrect pages.

For example, assume that your Web site displays information on the temperatures in major cities around the world. You use VuGen to create a Vuser script that accesses your Web site.

The Vuser accesses the site and executes a text check on this web page. For example, if the word **Temperature** appears on the page, the check passes. If **Temperature** does not appear because, for example, the correct page was not returned by the server, the check fails. Note that the text check step appears before the URL step. This is because VuGen registers, or prepares in advance, the search information relevant for the next step. When you run the Vuser script, VuGen conducts the check on the web page that follows.

Although the server may display the correct page when you record the script and when a single Vuser executes the script, it is possible that the correct page will not be returned when the server is under the load of many Vusers. The server may then be overloaded and may therefore return meaningless or incorrect HTML code. Alternatively, in some instances when a server is overloaded, the server returns a **500 Server Error** page. In both of these cases, you can insert a check to determine whether or not the correct page is returned by the server.

Note: Web checks increase the work of a Vuser, and therefore you may be able to run fewer Vusers per load generator. You should use Web checks only where experience has shown that the server sometimes returns an incorrect page.

To add an image or text check, see "How to Add Text Checks and Image Checks" on page 678.

Understanding Web Check Functions

When you add a text check, VuGen adds a **web_reg_find** function to your script. This function registers a search for a text string on an HTML page. Registration means that it does not execute the search immediately—it performs the check only after executing the next Action function, such as **web_url**.

Note: If you are working with a concurrent functions group, the **web_reg_find** function is executed only at the end of the grouping.

In the following example, the **web_reg_find** function searches for the text string "Welcome". If the string is not found, the next action function fails and the script execution stops.

```
web_reg_find("Text=Welcome", "Fail=Found", LAST);
web_url("Step", "URL=...", LAST);
```

In addition to the **web_reg_find** function, you can use other functions to search for text within an HTML page:

Several additional functions can be used for searching for text:

- web_find
- web_global_verification

The **web_find** function, primarily used for backward compatibility, differs from the **web_reg_find** function in that **web_find** is limited to HTML-based scripts (see **Recording Options > Recording** tab). It also has less attributes, such as instance, allowing you to determine the number of times the text appeared. When performing a standard text search, **web_reg_find** is the preferred function.

The **web_global_verification** function allows you to search the data of an entire business process. In contrast to **web_reg_find**, which only applies to the next Action function, this function applies to **all** subsequent Action functions such **web_url**. By default, the scope of the search is NORESOURCE, searching only the HTML body, excluding headers and resources.

The **web_global_verification** function is ideal for detecting application level errors that are not included the HTTP status codes. This function is not limited to an HTML-Based script (see **Recording Options > Recording** tab).

Data Format Extensions

VuGen supports the recording of many different types of data. New formats are constantly being created and VuGen must adapt to support them. Some formats are proprietary and use custom serialization, making it difficult for the user to understand the code due to binary and unformatted data. VuGen has developed a way to use Data Format Extensions (DFEs) to convert the code to more readable formats allowing you to parameterize and correlate this data.

Data Format Extensions work in ordered chains. Chains are comprised of extensions. Each HTTP message has only one **Body and Query String** section and therefore users can select the default chain for that section only, whereas HTTP messages can contain multiple **Headers and Cookies** sections. The user therefore can assign a different chain to each **Headers and Cookies** section in their message. For VuGen to correctly match the chain to the **Headers or Cookies** section, the name in the Name column must match the name of the **Headers or Cookies** section.

VuGen attempts to de-serialize the data using the DFEs in a chain one at a time. The extensions within a chain are each designed to de-serialize/serialize a specific data format. If an extension does not convert the specified data, this data is passed to the next extension in the chain. If an extension does successfully convert the data, you can configure VuGen to either continue to run the rest of the DFE chain on the converted data, or to end the process.

VuGen provides a number of built-in Data Format Extensions. For specific details about each of these DFEs, see "Data Format Extension List" on page 683.

Additionally, advanced users can manually add and modify chains, as well as create new Data Format Extensions.

For more information, see the *HP LoadRunner Data Format Extensions Developer Guide*.

Google Web Toolkit - Data Format Extension (GWT-DFE) - Overview

DFE (Data Format Extension) allows easier scripting of modern Web applications by providing the ability to decode and encode formatted data exchanged between the client and server. By providing the decoded format of the data, the information is presented in a readable format that allows for easier correlation and parameterization of the script.

GWT DFE is an addition to the already supported formats of Base64, JSON, URL Encoding, XML and Prefix-Postfix.

Key Features

- Format the GWT payload data into a readable form that enables you correlate and parameterize data.
- The formatted structure presents the names of the data fields.
- GWT- DFE provides automatic solution for GWT specific (STRONG_NAME_HEADER) correlations.

  For details, see "How to use GWT-DFE" on page 682.

Additional Information

GWT-DFE decodes a GWT- RPC (Remote Procedure Call) by converting the request/response payloads into XML based structures.

Web Snapshots - Overview

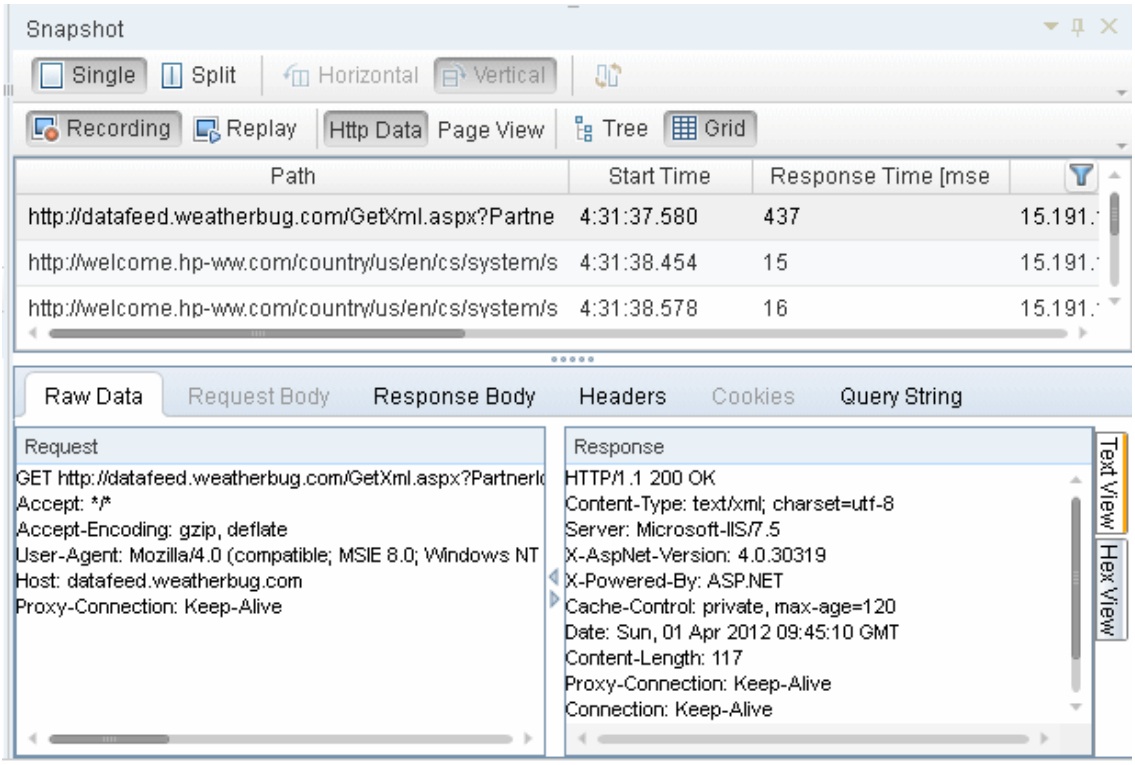
Vuser scripts based on the Web (HTTP/HTML) and Web (Click & Script) protocols utilize VuGen's Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

Web (HTTP/HTML) Snapshots

The snapshots show detailed information about some of the steps in the Vuser script. Each snapshot can be displayed using either the **Page** view or the more detailed **HTTP Data** view.

The HTTP Data view displays each HTTP transaction in either a tree view or a grid view. The transaction data is broken up into response data, request data, headers, cookies, and query strings.



Data in the snapshots can be displayed in a number of formats: Data view, Text view, and Hex view.

You can split the Snapshot pane to display two snapshots - typically a record snapshot and the corresponding replay snapshot. If both snapshots are displayed in the HTTP Data view, you can click the Sync button on the Snapshot pane toolbar to synchronize the data that is displayed in the two snapshots. For details, see "How to Work with Snapshots" on page 57.

Correlations and parameters can be created on response data by selecting the desired text and right-clicking.

For data that is difficult to work with (such as binary data) VuGen offers a variety of Data Format Extensions that can transform certain data types into more readable formats. Data that has been formatted by a Data Format Extension can be displayed in its original or formatted state. For more information, see "Data Format Extensions" on page 675.

XML Pages

VuGen supports record and replay for XML code within Web pages.

The XML code can appear in the script as a regular URL step or as a custom request. VuGen detects the HTML and allows you to view each document type definition (DTD), its entities, and its attributes. VuGen can interpret the XML when the MIME type displayed in the **RecContentType** attribute or the MIME type returned by the server during replay, ends with **xml**, such as **application/xml** or **text/xml**. The DTD is color coded, allowing you to identify each one of the elements. You can also expand and collapse the tree view of the DTD.

When you expand the DTD, you can parameterize the attribute values. You can also save the values in order to perform correlation using the standard correlation functions. For more information about the correlation functions, see the Function Reference (**Help > Function Reference**).

Note: VuGen cannot display a DTD with **XML islands**, segments of XML embedded inside an HTML page. VuGen displays only pages that are entirely XML.

How to Add Text Checks and Image Checks

There are a number of different types of checks that VuGen can add to your Web Vuser scripts. For background information, see "Text and Image Verification" on page 673.

How to Add a Text Check While Recording

1. In the web browser, select the desired text.
2. Click the **Insert Text Check** button  on the VuGen Recording toolbar. VuGen adds a **web_reg_find** function to the script.

How to Add a Text Check After Recording

1. In the Snapshot pane, display a snapshot that contains the text you want to verify.
2. In the Snapshot pane toolbar, click **HTTP Data** to display the HTTP Data view of the snapshot.
3. In the snapshot, select the text you want to verify. The text must be located in a response section of the snapshot - not in a request section.
4. Right-click and select **Add Text Check Step** from the menu.
5. Modify the options in the Find Text dialog box. For details on the dialog box options, press F1 when in the dialog box to open the Function Reference.
6. Click **OK** to insert the function into the Vuser script.

How to Add Other Text Checks After Recording

1. In the script editor, locate the position where you want to insert the check.
2. Select **View > Toolbox** to open the Toolbox.
   a. To insert a **web_reg_find** function, in the Toolbox, under **Services**, select **web_reg_find**.
   b. To insert a **web_global_verification** function, in the Toolbox, under **Services**, select the required **web_global_verification** function.
3. Drag the selected function to the required location in the Editor.
4. Enter the required details in the dialog box that opens. For details on the dialog box options, press F1 when in the dialog box to open the Function Reference.
5. Click **OK** to insert the function into the Vuser script.

How to Add an Image Check After Recording

1. In the Editor, locate the position where you want to insert the check.

2. Select **View > Toolbox** to open the Toolbox.
3. In the Toolbox, under **Web Checks**, select **web_image_check**.
4. Drag the selected **web_image_check** function to the required location in the Editor.
5. Enter the required details in the Image Check Properties dialog box. For details on the dialog box options, press F1 when in the dialog box to open the Function Reference.
6. Click **OK** to insert the function into the Vuser script.

How to Convert Web Vuser Scripts into Java

VuGen provides a utility that enables you to convert a script created for a Web Vuser into a script for Java Vusers. This also allows you to create a hybrid Vuser script for both Web and Java.

Convert a Web Vuser Script into a Java Vuser Script

1. Create an empty Java Vuser script and save it.
2. Create an empty Web Vuser script and save it.
3. Record a Web session using standard HTML/HTTP recording.
4. Replay the Web Vuser script. When it replays correctly, cut and paste the entire script into a text document and save it as a text **.txt** file. In the text file, modify any parameter braces from the Web type, "`{ }`" to the Java type, "`< >`".
5. Open a DOS command window and go to your product's **dat** folder.
6. Type the following command:

   ```
   <application_directory>\bin\sed -f web_to_java.sed filename >
   outputfilename
   ```

   where **filename** is the full path and filename of the text file you saved earlier, and **outputfilename** is the full path and filename of the output file.
7. Open the output file, and copy its contents into your Java Vuser script action section at the desired location. If you are pasting the contents into an empty custom Java template (Java Vuser type), modify the line containing `public int action()` as follows:

   ```
   public int action() throws Throwable
   ```

   This change is done automatically for recorded Java users (RMI and CORBA).
8. Parameterize and correlate the Vuser script as you would with an ordinary Java script, and run it.

How to Insert Caching Functions

This task describes how to use caching functions. Caching functions allow you to save stored data into your browser's cache, and load it at a later point in the script. For more information, see "Working with Cache Data" on page 673. The following steps describe how to use the caching functions.

1. Insert the **web_dump_cache** function into your script.
2. Run the script at least once.
3. Insert the **web_load_cache** function into your script before the Vuser actions.
4. Comment out the **web_dump_cache** function.
5. Run and save the script.

Example:

The following example illustrates a PeopleSoft Enterprise Vuser viewing the details of his paycheck.

```
Action()
{
//    web_add_cookie("storedCookieCheck=true; domain=pbntas05;
path=/");

web_load_cache("ActionLoad","FileName=c:\\temp\\{VuserName}paycheck",
LAST);
    web_browser("signon.html",
        DESCRIPTION,
        ACTION,
        "Navigate=http://pbntas05:8200/ps/signon.html",
        LAST);
    lr_think_time(35);
    web_edit_field("userid",
        "Snapshot=t1.inf",
        DESCRIPTION,
        "Type=text",
        "Name=userid",
        ACTION,
        "SetValue={VuserName}",
        LAST);

    web_edit_field("pwd",
        "Snapshot=t2.inf",
        DESCRIPTION,
        "Type=password",
        "Name=pwd",
        ACTION,
        "SetValue=HCRUSA_KU0007",
        LAST);
    lr_start_transaction("login");
        web_button("Sign In",
            "Snapshot=t3.inf",
            DESCRIPTION,
            "Type=submit",
            "Tag=INPUT",
            "Value=Sign In",
            LAST);
```

```
    lr_end_transaction("login", LR_AUTO);
    web_image_link("CO_EMPLOYEE_SELF_SERVICE",
        "Snapshot=t4.inf",
        DESCRIPTION,
        "Alt=",
        "Name=CO_EMPLOYEE_SELF_SERVICE",
        "Ordinal=1",
        ACTION,
        "ClickCoordinate=10,10",
        LAST); ...
    web_text_link("Sign out",
        "Snapshot=t7.inf",
        DESCRIPTION,
        "Text=Sign out",
        "FrameName=UniversalHeader",
        ACTION,
        "UserAction=Click",
        LAST);
/*web_dump_cache("paycheck","FileName=c:\\{VuserName}paycheck",
"Replace=yes", LAST);*/
    return 0;
}
```

How to Create or Modify Chains of Data Format Extensions

1. **Add or Select a Chain From the Chains Pane**

   Go to **Tools** > **Recording Options** > **Data Format Extension** > **Chain Configuration** node.

   For details, see "Data Format Extension > Chain Configuration Node" on page 281.

2. **Add or Modify the Data Format Extensions for the Selected Chain from the Chain: <chain name> pane**

   After you add an extension to the chain, select the appropriate option from the **Continue Processing** drop-down list.

   For details on the **Chain: <chain name>** pane, see "Data Format Extension > Chain Configuration Node" on page 281.

   If you add the **Prefix Postfix Extension** to your chain, the **Add Prefix/Postfix to Chain** dialog box opens, enabling you to configure the extension.

   For more details, see the **Edit DFE** row in "Data Format Extension > Chain Configuration Node" on page 281.

How to Apply Chains to Sections of the HTTP Message

1. **Set the Code Generation Configuration**

After selecting the **Enable data format extension** check box, select the **Format** option you want for your code.

You can apply extensions to the code and snapshots or snapshots only. You can also instruct to verify that your code was converted accurately.

For more details, see "Data Format Extension > Code Generation Node" on page 284.

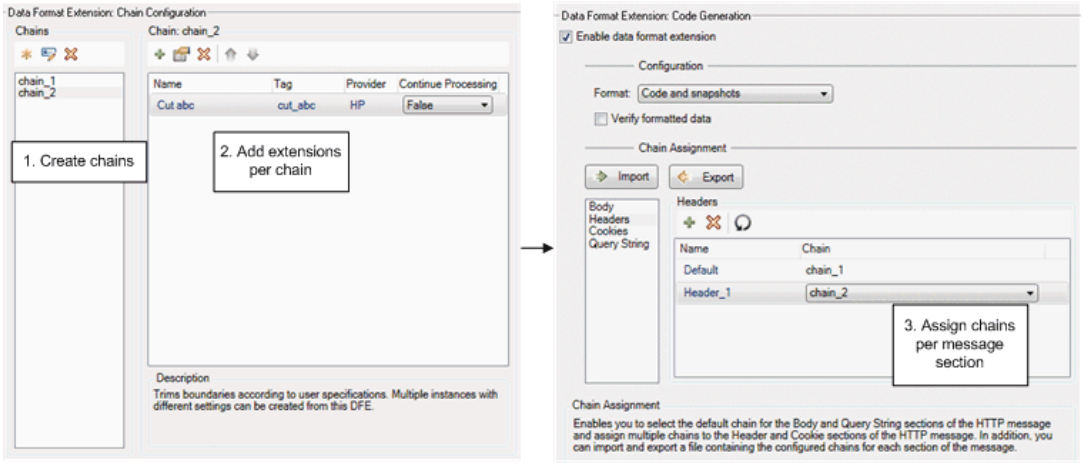
2. **Apply Chains to Sections of the HTTP Message**

Select a section of the HTTP message from the list of sections (Body, Headers, Cookies, Query String).

In the <**message section**> pane, add or modify the chains for the selected message section. You can modify only the default chain for the **Body** and **Query String** sections. You can add multiple chains for the **Headers** and **Cookies** sections.

For VuGen to correctly match the chain to the Headers or Cookies sections, the name in the **Name** column must match the name of the Headers or Cookies section in the message.

For more details, see "Data Format Extension > Code Generation Node" on page 284.



How to use GWT-DFE

This tasks describes how to create and assign a GWT-DFE chain.

Prerequisites

- LoadRunner includes JRE 1.5. However, many applications are compiled for JVM1.6 and higher. If your application is compiled with JDK1.6 or higher, replace the **<LoadRunner>\jre** directory with **<Program Files>\Java\jre6** before recording a GWT Application .
- To use the GWT DFE, you must have access to the GWT WAR folder as used by your development team which includes the following file types:
  - *.gwt.rpc files
  - *.jar files
  - *.class files

Configure GWT-DFE Headers

1. Create a script using the Web (HTTP/HTML) protocol.
2. Select **Record > Recording Options > HTTP Properties > Advanced > Headers**
3. Select **Record headers in list**
4. Select **x-gwt-permutation header**check box.

Create and Apply GWT-DFE Extension

1. Select **Recording Options > Data Format Extensions > Chain Configuration.**

   Click  to create and name a chain.

   Click  to add a GWT extension.

   Specify the Application Classpath

   In the **Add GWT to Chain** dialog box, edit the GWT DFE settings and set the classpath entries to include:

   - Click the  button to add the directory that contains.gwt.rpc files
   - Click the  button to add the application **classes** directory
   - Click the  button to add the application **JAR** files from the **WEB-INF\lib** directory.

   If the WAR file is extracted on the server, the directory structure will be as follows:

   ```
   <Server-
   Dir>\<applicationDir>\<MyGWTApplication>\<SomeDirContaining
   .gwt.rpc file>

   WEB-INF\classes

   WEB-INF\lib\gwt-servlet.jar

   WEB-INF\lib\gwt-servlet-deps.jar

   WEB-INF\lib\log4j.jar

   WEB-INF\lib\<AdditionalAUTRelatedJarFile>.jar
   ```

2. Select **Recording Options > Data Format Extension> Code Generation**

   Select **Enable Data Format Extension**check box.

   In the **Configuration** section, select **Code and snapshots** from **Format** drop-down list.

   Under the **Chain Assignment**, select **Body** and assign the chain, select **Headers** and assign the same chain.

Data Format Extension List

The following table lists the Data Format Extensions that come built in to VuGen. For more information about Data Format Extensions, see "Data Format Extensions" on page 675.

Data Format Extension	Description
Base64 Extension	Decodes strings that are encoded with as BASE64 encoder.
GWT Extension	Transforms GWT data to XML format.
URL Encoding Extension	Decodes strings that are encoded with URL encoding format.
JSON Extension	Transforms JSON data to XML format.
XML Validator Extension	Receives data and checks to see if it conforms with XML syntax. This check allows VuGen to perform correlations based on XPath and to display snapshot data in an XML viewer.
Prefix Postfix Extension	Enables you to cut data from the beginning and/or end of a string which you do not want decoded. You can add and customize as many prefix/postfix extensions as required. Each postfix/prefix extension created should have a unique display name and tag name.
Binary XML Extension	Transforms Microsoft WCF binary XML into XML format.
Remedy Extension	Transforms Remedy request data into XML format. Note that this extension does not transform Remedy response data - which is JavaScript code.
XSS Extension	Enables you to test sites that use Cross Site Scripting (XSS) defense code.

Web Services

Web Services - Adding Script Content

Web Service Testing Overview

SOA systems are based on Web Services, self-contained applications that can run across the Internet on a variety of platforms. The services are built using Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP). They serve as building blocks, thereby enabling the rapid development and deployment of new applications.

Using VuGen, you create Vuser scripts for testing your SOA environment. You can use a test generation wizard to automatically generate scripts, or you can create the scripts manually.

Adding Web Service Script Content - Overview

Web Services scripts let you test your environment by emulating Web Service clients.

After creating an empty Web Services script, as described in "Create a New Script Dialog Box" on page 120, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

Recording a Web Services Script

By recording a Web Services session, you capture the events of a typical business process. If you have already built a client that interacts with the Web Service, you can record all of the actions that the client performs. The resulting script emulates the operations of your Web Service client. After recording, you can add more Web Service calls and make other enhancements to the script.

When you record an application, you can record it with or without a Web Service WSDL file. If you include a WSDL file, VuGen allows you to create a script by selecting the desired methods and entering values for their arguments.VuGen creates a descriptive script that can be updated when there are changes in the WSDL.

If you record a script without previously importing a service (not recommended) VuGen creates SOAP requests instead of Web Service call steps. SOAP request arguments are less intuitive and harder to maintain.

For more information, see "How to Add Content" on page 692.

Adding New Web Service Calls

You can create a Web Services script by manually adding Web Service calls. You design the call based on operation, transport, arguments, and other properties.

For more information, see "How to Add Content" on page 692.

Importing SOAP Requests

VuGen lets you create Web Service calls from SOAP files. If you have a SOAP request file, you can load it directly into your script. VuGen imports the entire SOAP request (excluding the security headers) with the argument values as they were defined in the XML elements. By importing the SOAP, you do not need to set argument values manually as in standard Web Service calls.

For example, suppose you have a SOAP request with the following elements:

```
- <soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  - <q1:AddAddr xmlns:q1="http://tempuri.org/AddrBook/message/">
            <Addr href="#id1" />
    </q1:AddAddr>
   - <q2:Addr id="id1" xsi:type="q2:Addr"
xmlns:q2="http://tempuri.org/AddrBook/type/">
        <name xsi:type="xsd:string">Tom Smith</name>
             <street xsi:type="xsd:string">15 Elm Street</street>
             <city xsi:type="xsd:string">Pheonix</city>
```

```
            <state xsi:type="xsd:string">AZ</state>
            <zip-code xsi:type="xsd:string">97432</zip-code>
            <phone-numbers href="#id2" />
            <birthday xsi:type="xsd:date">1983-04-22</birthday>
      </q2:Addr>
...
```

When you import the SOAP request, VuGen imports all of the values to the Web Service call. To view the values, in the Step Navigator, right-click the step and then click **Show Arguments**.

To create a new Web Service call based on a SOAP request, you must first import a WSDL file. If a WSDL is not available, or if you want to send the SOAP traffic directly, you can create a SOAP Request step. You specify the URL of the server, the SOAP action, and the response parameter.

In the Editor, the SOAP Request step appears as a **soap_request** function, described in the Function Reference (**Help > Function Reference**).

For more information, see "How to Add Content" on page 692.

Analyzing Server Traffic

The main focus when testing enterprises and complex systems, is to measure the performance from the client end. Ordinarily, VuGen records the actions you perform in the application or browser, and generates a script emulating the client actions and requests to the server.

In certain test environments, you may be unable to record the client application to retrieve the requests to the server. This may be a result of the server acting as a client, or because you do not have access to the client application. In these cases, you can create a script using VuGen's **Analyze Traffic** feature.

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server.

For more information, see "How to Create a Script by Analyzing Traffic" on page 695.

Script Integration

You can use the completed script to test your system in several ways:

- **Functional Testing.** Run the script to see if your Web services are functional. You can also check to see if the Web service generated the expected values. For more information, see "Web Services - Preparing Scripts for Replay" on page 711.
- **Load Testing.** Integrate the script into a LoadRunner Controller scenario to test the performance of your system under load. For more information, see the *HP Controller* documentation.
- **Production Testing.** Check your Web service's performance over time through a Business Process Monitor configuration. For more information, see the *HP Business Process Monitor* documentation.

Web Service Call Attachments

When transferring binary files such as images over SOAP, the data must be serialized into XML. Serialization and deserialization can cause a significant amount of overhead. Therefore, it is common to send large binary files using an attachments mechanism. This keeps the binary data intact, reducing the parsing overhead.

Using attachments, the original data is sent outside the SOAP envelope, eliminating the need to serialize the data into XML and making the transfer of the data more efficient.

The formats used for passing a SOAP message together with binary data are MIME (Multipurpose Internet Mail Extensions) and the newer, more efficient DIME (Direct Internet Message Encapsulation) specifications. VuGen supports DIME for all toolkits, but MIME only for the Axis toolkit. To use MIME attachments for the .NET toolkit, see " User Handler Examples" on page 723.

VuGen supports the sending and receiving of attachments with SOAP messages. You can send Input (Request) or save Output (Response) attachments. For task details, see "How to Add Content" on page 692.

Output attachments are used to save the response as an attachment. You can choose one of the following options: **Save All Attachments** or **Save Attachment by Index**.

When you specify **Save All Attachments,** VuGen creates three parameters for each attachment based on the parameter name that your specify: a parameter containing the attachment data, the content type of the attachment, and a unique ID for the attachment.

For example, if you specify the name **MyParam** in the **Content** field, the parameter names for the first attachment would be:

```
MyParam_1
MyParam_1_ContentType
MyParam_1_ContentID
```

When you specify **Save Attachments by Index**, you specify the index number and name of the parameter in which to store the attachment. The parameter name that you specify for **Content**, is used as a prefix for the Content type and Content ID parameters.

Special Argument Types

VuGen handles special argument types such as derived, recursive, choice, and optional elements.

Derived Types

VuGen supports WSDLs with derived types. When setting the properties for a Web Service Call, VuGen allows you to use the base type or derived type for the argument. After you select a type, VuGen updates the argument tree node to reflect the new type. For details, see "New Web Service Call Dialog Box" on page 699.

Abstract Types

Abstract is a declaration type declared by the programmer. When an element or type is declared to be **abstract**, it cannot be used in an instance document. Instead, a member of the element's substitution group, provided by the XML schema, must appear in the instance document. In such a

case, all instances of that element must use the **xsi:type** to indicate a derived type that is not abstract.

When VuGen encounters an Abstract type, it cannot create an abstract class and replay will fail. In this case, VuGen displays a warning message beneath the **Type** box, instructing you to replace the Abstract type with a derived type.

Optional Elements

In WSDL files, optional parameters are defined by one of the following attributes:

```
minoccurs='0'
```

```
nillable='true'
```

minoccurs = 0 indicates a truly optional element, that can be omitted. Nillable means that the element can be present without its normal content, provided that the nillable attribute is set to true or 1. By default, the **minoccurs** and **maxoccurs** attributes are set to 1.

In the following example, **name** is mandatory, **age** is optional, and **phone** is nillable.

```
<s:element minOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" name="age" type="s:int" />
<s:element minOccurs="1" name="phone" nillable="true" type="s:string"
/>
```

The following table indicates the availability of the options:

Parameter type	Nil radio button	Include arguments in call
Mandatory	disabled	disabled
MinOccurs=0	disabled	enabled
Nillable	enabled	disabled

To include a specific optional argument in the service call, click the node and select **Include Argument in Call**. The nodes for all included arguments are colored in blue. Arguments that are not included are colored in gray.

If you include an element on a parent level, it automatically includes all mandatory and nillable children elements beneath it. If it is a child element, then it automatically includes the parent element and all other mandatory or nillable elements on that level. If you specify **Generate auto-value** to a parent element, VuGen provides values for those child elements that are included beneath the parent.

Note: VuGen interprets whether elements are mandatory or optional through the toolkit implementation. This may not always be consistent with the element's attributes in the WSDL file.

Choice Optional Elements

A Choice element in a WSDL defines a set of elements where only one of them appears in the SOAP message. In some cases, one of the Choice elements is optional, while the others are not. You can select the Choice element and still prevent its optional element from appearing in the

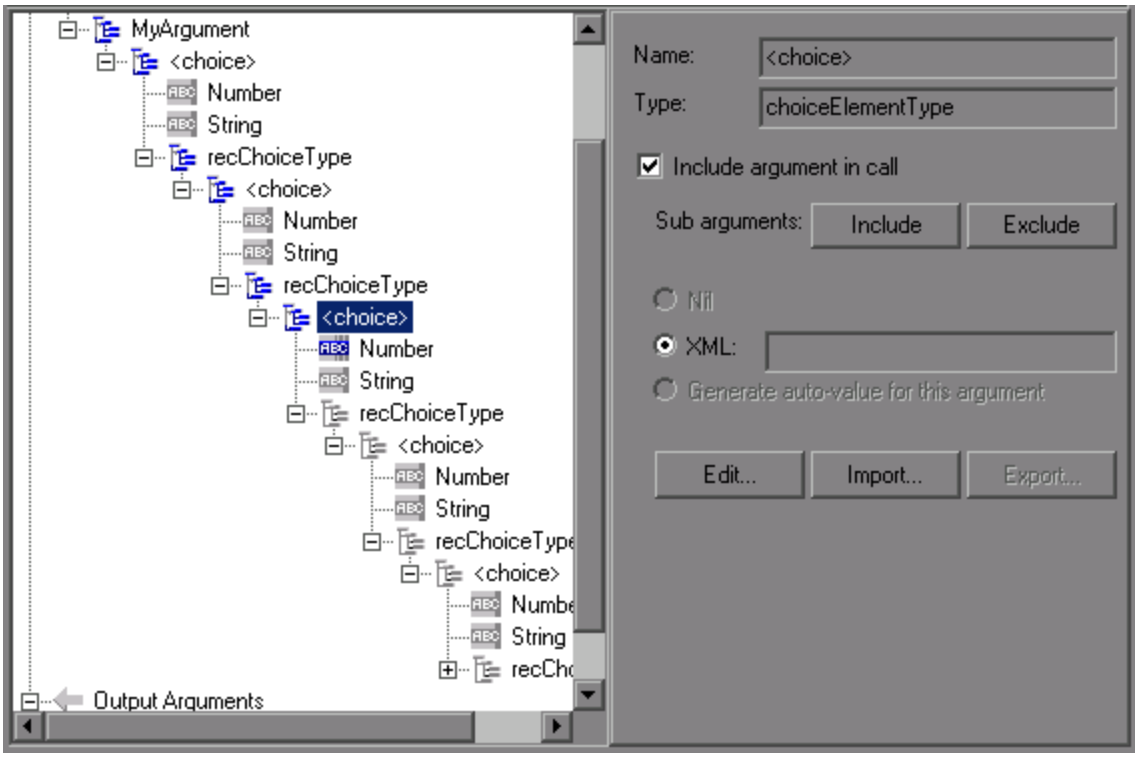
SOAP envelope. In the **Step Navigator**, select the Choice element, and clear the **Include argument in call** option. In Script view, delete the line that defines the Choice argument.

Recursive Elements

Using the Properties dialog box, you can control the level of recursive elements to include in the Web Service call.

To exclude a certain level and exclude those below, select the lowest parent node that you want to include and select **Include Argument in Call**. VuGen includes the selected nodes, its mandatory children, and all of its parent nodes.

In the following example, three levels of the Choice argument are included—the rest are not. Excluded nodes are grayed out.



Base 64 Arguments

Base 64 encoding is an encoding method used to represent binary data as ASCII text. Since SOAP envelopes are plain text, you can use this encoding to represent binary data as text within SOAP envelopes.

When VuGen detects a WSDL element of **base64Binary** type, it lets you provide an encoded value. You can specify a value in two ways:

- **Get from file.** Reference a file name.
- **Embed encoded text.** Specify the text to encode.

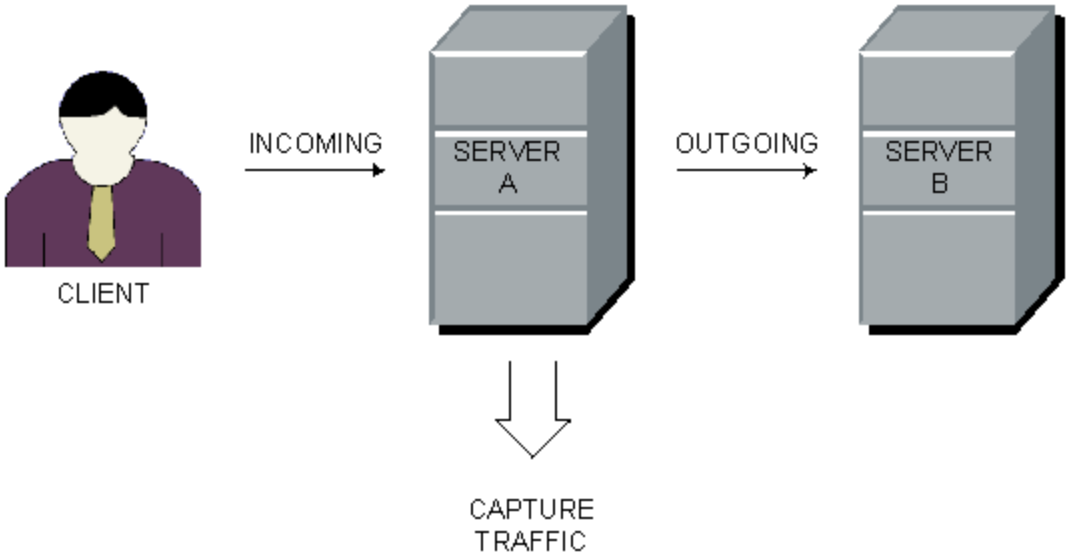
For details, see "Process Base64 Data - Simple Data Dialog Box" on page 706.

Server Traffic Scripts Overview

The main focus when testing enterprises and complex systems, is to measure the performance from the client end. Ordinarily, VuGen records the actions you perform in the application or browser, and generates a script emulating the client actions and requests to the server.

In certain test environments, you may be unable to record the client application to retrieve the requests to the server. This may be a result of the server acting as a client, or because you do not have access to the client application. In these cases, you can create a script using VuGen's **Analyze Traffic** feature.

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server. The steps in creating a script by analyzing server traffic are described in "How to Create a Script by Analyzing Traffic" on page 695.



There are two types of emulations: **Incomingtraffic** and **Outgoing traffic**.

Incoming traffic scripts emulate situations in which you want to send requests to the server, but you do not have access to the client application, for example, due to security constraints. The most accurate solution in this case is to generate a script from the traffic going **into** the server, from the side of the client.

When you specify an Incoming server network traffic, you indicate the IP address of the server and the port number upon which the application is running. VuGen examines all of the traffic going into the server, extracts the relevant messages, and creates a script. In the above diagram, if the client is unavailable, you could create an Incoming script to emulate the requests coming into **Server A**.

Outgoing Traffic scripts emulate the server acting as a client for another server. In an application server that contains several internal servers, you may want to emulate communication between server machines, for example between **Server A** and **Server B** in the above diagram. The solution in this case is to generate a script from the traffic sent as output **from** a particular server.

When you create an Outgoing traffic script, you indicate the IP address of the server whose outgoing traffic you want to emulate, and VuGen extracts the traffic going out of that server. In the

above diagram, an Outgoing script could emulate the requests that **Server A** submits to the **Server B**.

- "Capture Files" below
- "Filtering Traffic" below
- "Data on Secure Servers" on next page

Capture Files

A capture file is a trace file containing a log of all TCP traffic over the network. Using a sniffer application, you obtain a dump of all of the network traffic. The sniffer captures all of the events on the network and saves them to a capture file.

To generate a smaller, more manageable script, try to capture the network traffic only for the time that you perform actions in your application.

> **Note:** Capture files do not contain loopback network traffic.

You can obtain a capture file using the command line utility or any existing capture tool.

The VuGen command line utility, **lrtcpdump**, is located in the product's **bin** folder. There is a separate utility for each of the platforms: **lrtcpdump.exe** (Windows), **lrtcpdump.hp9**, **lrtcpdump.ibm**, **lrtcpdump.linux**, and **lrtcpdump.solv4**.

External Capture Tools

Most UNIX operating systems have a built-in version of a capture tool. In addition, there are many downloadable capture tools such as **Ethereal/tcpdump**.

When using external tools, make sure that all packet data is being captured and none of it is being truncated.

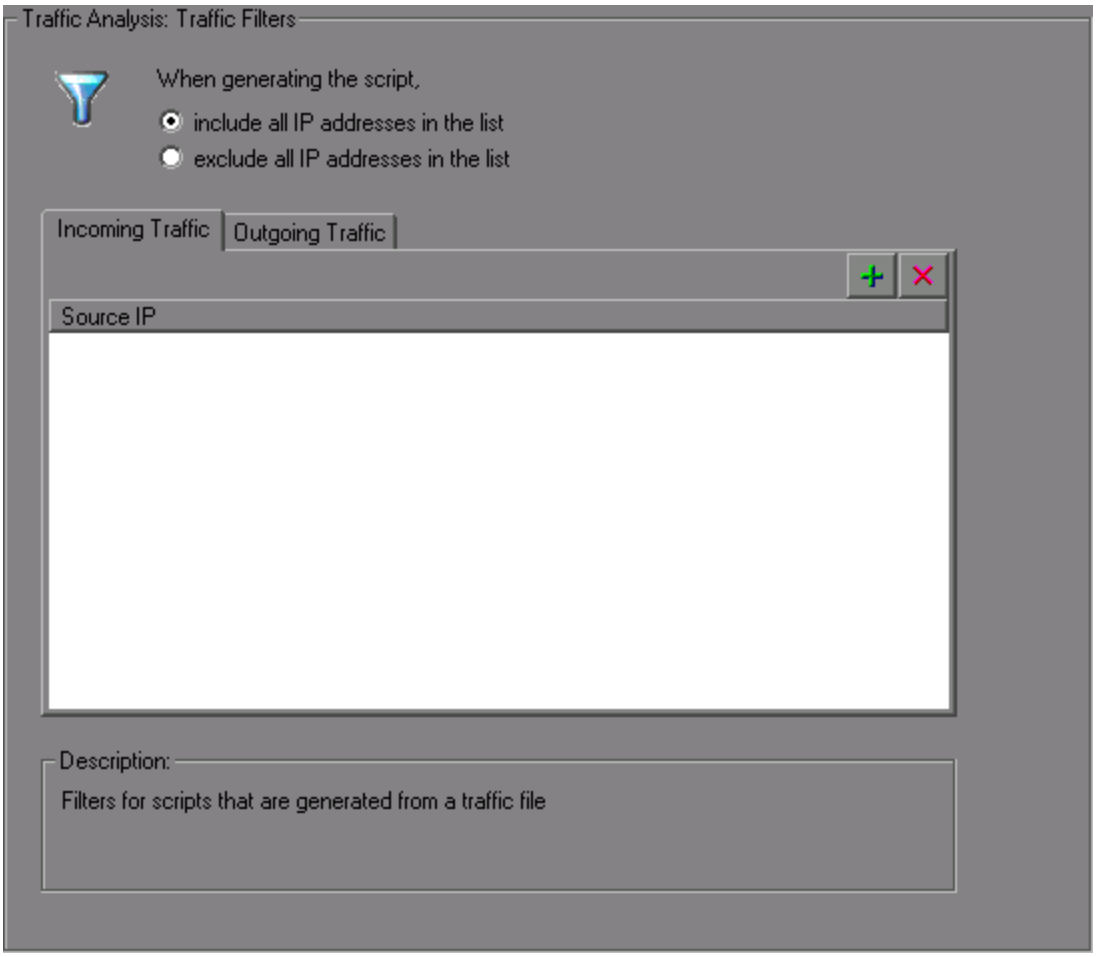
Certain capture utilities require additional arguments. For example, **tcpdump** requires the **-s 0** argument in order to capture the packets without truncating their data.

Filtering Traffic

You can provide a filter to drill down on specific requests going to or from a server, by specifying its IP address and port.

> **Tip:** Several external capture tools allow you to filter the IP addresses while capturing the traffic.

You filter the requests by choosing the relevant host IP addresses. The filter can be inclusive or exclusive—you can include only those IPs in the list, or include everything except for those IPs that appear in the list.



For more information, see "How to Create a Script by Analyzing Traffic" on page 695.

Data on Secure Servers

To analyze traffic from a secure server, you must provide a certificate containing the private key of the server.

If the traffic is SSL encrypted, you must supply a certificate file and password for decryption. If you want traffic from multiple servers to be reflected in the script, you must supply a separate certificate and password for each IP address that uses SSL.

For more information, see "How to Create a Script by Analyzing Traffic" on page 695.

How to Add Content

This topic describes how to add content, such as Web Service calls, to a Web Services Vuser script.

Prerequisites

Create an empty Web Services Vuser script. Click **File > New Script and Solution** and choose the **Web Services** protocol. You can create either a single-protocol or multi-protocol Vuser script.

Record a Session - Optional

1. Click the **Start Recording** button  on the VuGen toolbar or press Ctrl+R to open the Recording Wizard > Specify Services screen.

   For user interface details, see "Specify Services Screen" on page 697.

2. Add services to the list. Click **Import** to load a WSDL for the test. Indicate the location of the WSDL file.

   For user interface details, see "Import Service Dialog Box" on page 755.

3. Click **Next**. Specify the location of the application and any other relevant arguments. See the "Specify Application to Record Dialog Box" on page 697.

Add a New Service Call - Optional

1. Import a service. Click **Manage Services** to access the Import dialog box.

   For user interface details, see "Import Service Dialog Box" on page 755.

2. Click the cursor at the desired location in your script (**Editor**) or in the test steps (**Step Navigator**).

3. Click the **Add Service Call** button. The New Web Service Call dialog box opens.

4. In the Select Web Service Call section, select a **Service**, **Port Name**, and **Operation**.

5. To specify an endpoint other than the default **Target Address**, select **Override Address** and insert the new endpoint to which you want to submit the requests.

6. Expand the nodes and specify argument values. To create sample values for all Input arguments, select the **Input Arguments** node and click **Generate**. To edit, import, or export the element's XML structure, see "How to Assign Values to XML Elements" on next page.

7. To parameterize an input argument, click the node and select the **Value** option. Click the ABC icon and proceed with parameterization. For more information, see "Parameters" on page 227.

8. Select the **Transport Layer Configuration** node to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see "How to Send Messages over JMS" on page 727.

Add Attachments - Optional

1. To add an attachment to an input argument, choose an operation in the left pane. Select **Add to request (Input)**. VuGen prompts you to enter information about the attachment and adds it to the method's tree structure. For details, see the "Add Input Attachment Dialog Box" on page 705.

2. To specify an output attachment through which to store output arguments, choose an operation in the left pane. Select **Save received (Output)**. Select the desired option: **Save All Attachments** or **Save Attachment by Index** based on their index number—beginning with 1. For details, see "Web Service Call Attachments" on page 687.

3. To edit the properties of either an Input or Output attachment, click the attachment in the left pane, and enter the required information in the right pane.

Specify SOAP Headers - Optional

Select the **CustomSOAP Header** node in the left pane and enable the **Use SOAP header** option.

You must individually specify SOAP headers for each element. To compose your own, click **Edit** and edit the XML. To import an XML file for the SOAP header, click **Import**.

Import SOAP - Optional

1. Import a service if one is available. Click **Manage Services** to access the Import dialog box. For more information, see the "Import Service Dialog Box" on page 755.
2. Click the **Import SOAP** button to open the Import SOAP dialog box.
3. Browse for the XML file that represents your SOAP request.
4. Select the type of step you would like to generate: **Create Web Service Call** or **Create SOAP Request**. In order to create a Web Service Call, you must first import at least one WSDL that describes the operation in the SOAP request file. To view the SOAP before loading it, click **View SOAP**.
5. Click **Load** to import the XML element values.

   For a **Web Service Call**, set the properties for the Service call as described in the "New Web Service Call Dialog Box" on page 699.

   For a **SOAP Request**, provide the URL and the other relevant parameters.
6. For a Web Service Call, if there are multiple services with same operation (method) names, select the service whose SOAP traffic you want to import.
7. Click **OK** to generate the new step within your script.

Analyze Server Traffic - Optional

To create a script by analyzing a file containing a dump of the server traffic, click **Analyze Traffic**.

For details, see "Server Traffic Scripts Overview" on page 690.

How to Assign Values to XML Elements

This task describes how to work with XML elements by manually editing the code, importing an external file, and exporting the XML for later use.

1. **Prerequisites**

   Import a service and create a new Web Service call. Alternatively, in the Step Navigator, right-click a step and select **Show Arguments**.
2. **Select the element**

   In the left pane, select a complex type or array argument. In the right pane, click **XML**. The XML field shows the XML code as a single string.
3. **Import a file - optional**

   To import a previously saved XML file, click **Import** and specify the file's location.
4. **Edit the XML elements - optional**

   To edit the XML structure and element values, click **Edit**. The XML Editor opens. To import a previously saved XML file, click **Import File.**

- To manually edit the code, click the **Text View** tab.
- To modify the XML through a graphical interface, click the **Step Navigator**. Use the shortcut menu to add children and sibling elements and rename the node. Click **Insert** from the shortcut menu to add a new element before or after the selected one.

5. **Export a file - optional**

   To save your XML data to a file so it can be used for other tests, click **Export** and specify a location.

How to Generate a Test Automatically

This task describes how to create requirements or tests for checking your service.

1. **Open the wizard**

   Select **File > New** to open the New Virtual User dialog box. Select **SOA Test Generator** in the left pane and click **Create**.

2. **Add a service**

   Proceed to the next screen and click **Add** to import at least one service. If your service is not ready yet, you can use an emulated service. For details, see "How to Add and Manage Services" on page 746. Click **Next**.

3. **Select testing aspects**

   Expand the nodes and select the desired testing aspects. Click **Next**.

4. **Specify a location**

   Specify a test name and a location for the test scripts: **HP ALM** or a **local file system**. If you specified ALM, click **Connect** to log on to the server and **Browse** to locate the test node.

5. **Complete the test generation**

   Review the summary and include or exclude any scripts from the generation. Click **Generate**.

6. **Open the scripts**

   In the final screen, review the list of generated scripts and indicate which ones to open. Click **Finish**.

How to Create a Script by Analyzing Traffic

This task describes how to create a script using a network traffic file.

1. **Create a capture file on a Windows Platform - optional**

   Create a capture file containing a log of all TCP traffic over the network on a Windows platform. Use a downloadable capture tool or use the tool provided in the product's **bin** folder, **lrtcpdump.**<*platform*>**.**

   a. Run the capture utility in a command window **lrtcpdump -f<***file_name***>.cap**. **lrtcpdump** prompts you to select a network card.

b. Type in the number of the interface card (if there are multiple ones.) and click **Enter**.

c. Perform typical actions within your application.

d. Return to the command window and click **Enter** to end the capture session.

e. Place the capture file on the network in a location accessible to the machine running VuGen.

2. **Create a capture file on a Linux Platform - optional**

Create a capture file containing all TCP traffic over the network on a Linux platform.

a. Locate the appropriate **lrtcpdump** utility for your platform in the product's **bin** folder. Copy it to a folder that is accessible to your Linux machine. For example, for an HP platform, copy **lrtcpdump.hp9**. If using FTP, make sure to use the binary transfer mode.

b. Switch to the root user to provide execution permissions: **chmod 755 lrtcpdump.<***platform***>**

c. If there are multiple interface cards, **lrtcpdump** uses the first one in alphabetical order. To get a complete list of the interfaces, use the **ifconfig** command.

d. Run the utility with its complete syntax, specifying the interface and file name. For example, **lrtcpdump.hp9 -ietho -f<***file_name***>.cap**. The capturing of the network traffic begins.

e. Perform typical actions within your application.

f. Return to the window running **lrtcpdump** and follow the instructions on the screen to end the capture session.

g. Place the capture file on the network in a location accessible to the machine running VuGen.

3. **Open the Analyzing Traffic wizard**

Click the **Analyze Traffic** button or select **Vuser > Analyze Traffic**. For details, see "Specify Services Screen" on next page.

4. **Import a Service - optional**

Add one or more services to the list (optional). Click **Import** to load a WSDL file. For details, see the "Import Service Dialog Box" on page 755.

Click **Next**.

5. **Specify traffic information**

Specify a capture file and the section of the script into which you want to load the traffic: **vuser_init**, **Action**, or **vuser_end**.

Indicate whether you want to analyze **Incoming** or **Outgoing** traffic. Specify the server whose traffic you want to analyze.

For details, see "Specify Application to Record Dialog Box" on next page.

6. **Filter the IP addresses - optional**

Click the **Filter Options** button to open the Recording options and indicate which IP addresses

to ignore or include.

For details, see "Recording Options" on page 258.

7. **Configure the SSL - optional**

   Click the **SSL Configuration** button to add SSL certificates. This is necessary in order to analyze traffic from a secure server.

   For details, see the "SSL Configuration Dialog Box" on page 710.

Specify Services Screen

This dialog box enables specify the basic details needed to begin recording a script.

To access	**VuGen > Start Record** button
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Details...	Opens the Manage Services dialog box for providing further information about the service. For more information, see "Manage Services Dialog Box" on page 751.
Import	Opens the Import Service dialog box. For more information, see the "Import Service Dialog Box" on page 755. For user interface details, see the "Import Service Dialog Box" on page 755.
Delete	Removes the selected service from the list.
<services list>	A list of the available services: • **Service Name.** The native name of the service. • **WSDL Location.** The source of the WSDL.

Specify Application to Record Dialog Box

This dialog box enables specify the basic details needed to begin recording a Web Services script.

To access	**VuGen > Start Record** button, **Next**
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
Advanced Options....	Opens the Recording Options dialog box. For user interface details, see "Recording Options" on page 258.

, continued

Record default Web browser	Records the default browser actions. Specify the starting URL or click the Browse button to navigate to a location.
Record any application	Records any Win32 application. You can also specify the following: • **Program to record.** Select the browser, internet application, or Win32 application to record • **Program arguments** (Win32 Applications only). Command line arguments for the application. For example, if you specify plus32.exe with the command line options peter@neptune, it connects the user Peter to the server Neptune when starting plus32.exe. • **Working directory.** A working folder for the application (only when required by the application).
Record into action	The section into which you want to record: **vuser_init**, **Action**, or **vuser_end**. For actions you want to repeat, use the **Action** section. For initialization steps, use **vuser_init**.
Record application startup	In the following instances, it may not be advisable to record the startup: • If you are recording multiple actions, in which case you need to perform the startup in only one action. • In cases where you want to navigate to a specific point in the application before starting to record. • If you are recording into an existing script.

Import SOAP Dialog Box

This dialog box enables you to create a test step based on a SOAP file.

To access	Use one of the following: • Click  • **SOA Tools > Import SOAP**
Relevant tasks	"How to Add Content" on page 692 "Adding Web Service Script Content - Overview" on page 685

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Browse.** Locate the XML file containing SOAP traffic.
	Loads the element values from the SOAP file.

, continued

Manage Services	Opens the Manage Services dialog box for importing and configuring services.
View SOAP	Opens the SOAP file in a browser for viewing.
<Call type>	The type of call to generate in the script/test: • **Web Service Call**. Requires the import of a service. • **SOAP Request.** Generates a soap_request step in the script.
SOAP Request Properties	The properties of the SOAP request (only visible for **SOAP Request** type calls). Specify the following: • **URL.** The URL or IP address of the server to which to submit the request. • **SOAP Action.** The SOAP action to include in the request (applicable if there are multiple actions). • **Response Parameter.** A parameter name to store the response of the SOAP or Web Service call request.

New Web Service Call Dialog Box

This dialog box lets you create and configure a new Web Service call.

To access	Open a Web Service Vuser script and then click **SOA Tools > Add Service Call** or click the **Add Service Call** button  on the VuGen toolbar.
Important Information	To access the Web Service call properties for existing Web Service calls, select a step in the **Step Navigator** and choose **Properties** from the shortcut menu.
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<service argument tree> (left pane)	An expandable tree hierarchy of the Service containing the following nodes: • <operation name> • Transport Layer Configuration • Custom SOAP Header • Input Arguments • Output Arguments

, continued

<parameter values> (right pane)	Enables you to set and select values for each of the left pane's nodes.
Select Web Service Call	Lets you set the following items: • **Service.** A dropdown list of all of the imported services, with the name derived from the WSDL. • **Port Name.** A dropdown list of available ports through which to send the request. • **Operation.** A dropdown list of the service's operations. • **Target Address**. The default endpoint of the service. • **Override Address.** Allows you to enter an alternate endpoint address in the **Target Address** box.

<Operation Name> Node

Allows you to generate sample values for the operation's input arguments and add attachments.

User interface elements are described below:

UI Element	Description
Method	The name of the selected operation (read-only).
Generate auto-values for input arguments	Automatically creates sample values for all of the input arguments, based on their data type.
Attachments	Handles input and output attachments: • **Add to Request (Input).** Attaches a file or parameter value to the request. • **Save Received (Output).** Saves the response to a parameter.
Step properties	Lists the following service call properties and their values: • WSDL file location • Service name • Port name • Target address • SOAP action • SOAP namespace

Transport Layer Configuration Node

User interface elements are described below:

UI Element	Description
HTTP/S Transport	Sets the transport method to HTTP or HTTPS transport.
Async Support	Marks the Web Service call as an asynchronous message activated by an event: **Async Event.** An arbitrary name for the event. **Note:** Add a **Web Service Wait For Event** step to the script, to instruct the replay engine to wait for the event.
WSA Support	Enables WS-Addressing. Use one of the following options for a reply: • **WS-A Reply.** An IP address of the server to reply to when the event occurs. • **Autodetect.** Reply to the current host when the event occurs. This is useful when running the same script on several machines. **Tip:** To use WS-Addressing calls in synchronous mode, leave the **Async Event** box empty. In Script view, remove the **AsyncEvent** argument. This instructs the replay to block script execution until the complete response is received from the server.
JMS Transport	Sets the transport method to JMS for *synchronous* messages. For details, see "Testing Web Service Transport Layers Overview" on page 711. **Note:** For JMS *asynchronous* messages, manually add a **JMS Send Message Queue** or **JMS Receive Message Queue** step to the script, to set up the message queue information.
Override JMS Queues	Enables you to provide the request and response queues.
Request Queue	The queue name for the request message.
Response Queue	The queue name for the response message.

Custom SOAP Header Node

Lets you specify additional application-generated header elements to include in the SOAP envelope of an HTTP message. For task details see "How to Add Content" on page 692.

User interface elements are described below:

UI Element	Description
Edit...	Opens an XML editor that lets you view and edit the SOAP header XML code.
Import...	Opens the **Select XML File to Import** dialog box.

, continued

Export...	Opens the **Export SOAP Header into File** dialog box.
	Opens the **Select or Create New Parameter** dialog box.
Use SOAP Header	Includes a SOAP header in the HTTP request.
Header	The header source: • For an imported file: Header element as it appears in the imported file. • For a parameter: The parameter name (in curly brackets)

Input Arguments Node

Lets you set the properties and generate values for all input arguments.

User interface elements are described below:

UI Element	Description
Include All	Includes all of the method's arguments in the Web Service call.
Reset	Resets the arguments to their original state. It removes their inclusion in the call, and sets them to the values in the WSDL.
Generate	Generates sample data for all of the input arguments.
Edit Argument	Opens the pane for editing the selected argument's value.
Name	The name of the operation (read-only)
Argument List	A list of the input arguments. •  Simple parameters •  Arrays (shows the top level only).

<Input Argument Name> Node

When selecting an input argument, the right pane allows you to specify argument values.

User interface elements are described below:

UI Element	Description
Add..	Opens the "Add Array Elements Dialog Box" on page 706 for adding a new array element to the input argument (only visible when selecting a parent node in the input array). For details, see the "Add Array Elements Dialog Box" on page 706.
Delete...	Removes the selected array element in the input argument (only visible when selecting a parent node in the input array).

, continued

Include	Includes the sub-arguments of the selected argument, in the Web Service call. This is only enabled for an argument with sub-arguments with the **Include argument in call** option enabled.
Exclude...	Excludes the sub-arguments of the parent argument, from the Web Service call.
Edit...	Opens an XML editor for editing the XML code containing the argument values. The only changes saved are the element values and the number of array elements.
Import...	Opens the **Select XML File to Import** dialog box,
Export...	Opens the **Export argument XML into file** dialog box.
	Opens the **Select or Create Parameter** dialog box.
Name	The name of the argument or array.
Include argument in call	Include the argument in the call. For arrays, click **Include** to add the sub arguments to the call. To exclude all omittable arguments, click **Exclude**.
Type	The argument type as defined in the WSDL. When the WSDL contains derived types, this box becomes a drop down list. For details, see "Special Argument Types" on page 687.
Nil	Sets the Nillable attribute to **true**.
XML (for arrays only)	• **XML.** Enables the **Edit**, **Import,** and **Export** buttons. By editing the XML, you can manually insert argument values. Click on the **ABC** icon to replace the entire XML structure with a single XML type parameter. **Note**: This import operation handles XML files that were previously exported—not standard SOAP files. • **Generate auto-value for this argument.** Inserts automatic values for all children elements. • **Add**/**Delete**. Adds or removes elements from the array.
Value (for non -array elements)	The argument value. To parameterize this value, click the abc icon (only available for non-arrays).
Generate auto-value for this argument	Generates a sample value for the selected argument.

Output Arguments Node

Lets you see the properties of all output arguments.

User interface elements are described below:

UI Element	Description
Edit Argument	Opens the pane for editing the selected argument's value.
Name	The name of the operation (read-only)
Negative Testing	**Enables Negative Testing.** Confirms that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued a SOAP Fault, and not a SOAP result response. Select an **Expected Response**. • **SOAP Result.** A SOAP response to the request. • **SOAP Fault.** A response indicating that the SOAP request was invalid. Negative Testing applies only to SOAP faults. • **HTTP Error.** An HTTP error, such as Page Not Found, unrelated to Web Services. For details, see Negative Testing Overview.
Argument List	A list of the output arguments and the corresponding parameters storing the values.

<Output Argument Name> Node

Lets you specify a parameter for storing the value of the output argument.

User interface elements are described below:

UI Element	Description
Add..	Opens the "Add Array Elements Dialog Box" on page 706 for adding a new array element to the output argument (only visible when selecting a parent node in the output array). For details, see the "Add Array Elements Dialog Box" on page 706.
Delete	Removes the selected array element in the output argument (only visible when selecting a parent node in the output array).
Name	The name of the output argument or array.
Save returned value in parameter	Saves the value of the selected argument to a parameter. To specify a custom parameter name, modify the default Param_<*arg_name*> in the **Parameter** field.
Nil	Sets the value of the current argument to **nil=true.**
XML (for arrays only)	XML code containing the argument values. To parameterize this value, click the abc icon (only available for arrays).

Output Attachments Node

Lets you set the properties for output attachment parameters. This is only visible if you enabled output attachments in the "New Web Service Call Dialog Box" on page 699.

User interface elements are described below:

UI Element	Description
Add..	Adds a new index-based output argument. This is available only when choosing **Save Attachments by Index**.
Save All Attachments	Saves all output attachments to a parameter with the following properties: • **Content.** An editable name for the parameter storing the attachment • **Content-type.** The type of parameter (read-only). • **Content-ID.** A unique ID for the parameter (read-only).
Save Attachments by Index	Saves the output attachments to index-based parameters. To set the index, select one of the parameters and modify the index number in the right pane.

<Output Argument> Node

Lets you set the properties for output attachment parameters. This is only visible if you enabled output attachments in the "New Web Service Call Dialog Box" on page 699.

User interface elements are described below:

UI Element	Description
Delete Attachment	Deletes the selected attachment parameter. If you have saved the attachments by index, it only removes the selected item.
Index	An index number for the parameter. This field is only enabled when you select **Save Attachments by Index** in the "New Web Service Call Dialog Box" on page 699.
Content	An editable name for the parameter storing the attachment
Content-type	The content type of parameter (read-only).
Content-ID	A unique content ID for the parameter (read-only).

Add Input Attachment Dialog Box

This page enables you to add input attachments to your Web requests.

To access	Click Add Service Call and select the top node—the Operation name. Select **Add to Request (input)** in the **Attachments** section.
Important information	You must import a service before adding an attachment to a service call. For background information, see "Web Service Call Attachments" on page 687.
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
Take data from	The location of the data. • **File.** The file location: ▪ Absolute Path: The full path of the file. Note that this file must be accessible from all machines running the script. ▪ Relative Path: (recommended) A file name. Using this method, during replay, VuGen searches for the attachment file in the script's folder. To add it to the script's folder, select **File > Add Files to Script** and specify the file name. • **Parameter.** The name of a parameter containing the data.
Content-type	The content type of the file containing the data. The **Detect Automatically** option instructs VuGen to automatically determine the content type. The **Value** box accepts manual entries and provides a dropdown list of common content types.
Content-id	A unique identifier for the attachment. By default, VuGen generates this automatically. Optionally, you can specify another ID in the **Value** box.

Add Array Elements Dialog Box

This page enables you to add elements to an argument array with an identical structure to the existing array. This is available for both Input and Output arguments.

For Input elements, you can base the new array's values on an existing element.

To access	Click  . Select an argument node that is an array.
Important information	You must have an array in your argument tree in order to view this dialog box.
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
Name	The name and index of the array's parent node.
Start Index	The index from which to add new array elements.
Elements	The number of identical array elements to add to your argument tree.
Copy values from index	Creates the new array elements with the values of a specific array element (only available for Input arguments).

Process Base64 Data - Simple Data Dialog Box

This dialog box enables you to set the encoding options for your simple base64 data.

To access	For simple, non-complex Base64 values: • Select an input argument in the Web Service Call Properties of **Base64** type. • Select the **Value** option. • Choose **Embed encoded text**. • Click the Browse button.
Important information	For a complex array, use the "Process Base64 Data - Complex Data Dialog Box" below.
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
Decode to File ...	Allows you to save the decoded text to a file.
Text to encode	For complex data, use the "Process Base64 Data - Complex Data Dialog Box" below.
Encoding Options	A list of encoding methods. **Default:** Unicode (UTF-8)
Encoded data	The encoded version of the data from the **Text to encode** pane.

Process Base64 Data - Complex Data Dialog Box

This dialog box enables you to set the encoding options for your complex base64 data.

To access	For complex Base64 values: • Select a complex input argument in the Web Service Call Properties, of **Base64** type. • Select the **Value** option click the Parameter icon. • Replace the value with a parameter. • Right-click the Parameter icon in the Value box and select **Parameter Properties**. • Click the **Edit Data** button. • In the desired values set column, click the B64 button.
Important information	For simple, non-complex data, use the "Process Base64 Data - Simple Data Dialog Box" on previous page.
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
Encode	Encodes the specified file.
Decode to File ...	Enables you to save decoded data to a file. This is usually data obtained during replay.
File	Encode the file by reference or its contents. • **File path.** The file to encode. • **Link to file.** References the file containing the values If cleared, it uses the content of the specified file. It copies the content to the script folder. **Tip.** For text exceeding 10KB, enable **Link to file**.
Text	Encodes the specified text string. • **Text to encode.** The Base64 text to encode. As you type the text, VuGen encodes it in the **Encoded data** pane. • **Encoding Options.** A list of encoding methods. The default is Unicode (UTF-8).

Aspects List

The following table lists the available testing aspects:

Aspect Name	Description
Positive Testing	Generates a full positive test that checks each operation of the service.
Standard Compliance	Checks the service's compliancy with industry standards such as WS-I and SOAP.
Service Interoperability	Tests the interoperability of the service's operations with all supported Web Services toolkits. Contains the following sub-aspects: • **.NET Framework.** Tests that the services are fully interoperable with .NET Framework WSE 2 Toolkit by calling all of its operations with default/expected values. • **Axis/Java Based Web Services.** Tests that the services are fully interoperable with Axis 1.3 Web Services Framework by calling all of its operations with default/expected values.
Security Testing	Tests service security. Contains the following sub-aspects: • **SQL Injection Vulnerability.** Checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters. • **Cross-site Scripting (XSS).** Attempts to hack the service by injecting code into a Web site that will disrupt its functionality.

, continued

Boundary Testing	Using the negative testing technique, creates tests to manipulate data, types, parameters, and the actual SOAP message to test the service to its limits. Contains the following sub-aspects: • **Extreme Values.** Provides invalid data types to the services and verifies they are not accepted. • **Null Values.** Provides NULL parameters to the services to verify they are not accepted.
Performance Testing	Contains the following sub-aspects: • **Stress Testing.** Tests the maximum load that can be placed on the application. • **Overload Sustainability Testing.** Tests how well the hardware allocated for the application can support the number of anticipated users. • **Volume Testing.** Tests that the system can handle a massive data entry. • **Longevity Test.** Tests that the system can sustain a consistent number of concurrent Vusers executing transactions using near-peak capacity, over a minimum 24-hour period. • **Scalability Testing.** Repeated stress, overload, volume, and longevity tests with different server or network hardware configurations.

Specify Services Screen

This wizard screen enables you to select Web services to associate with your traffic-based script.

To access	**Analyze Traffic** button
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below (unlabeled elements are shown in angle brackets):

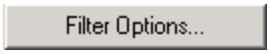
UI Element	Description
Details...	Opens the **Manage Services** dialog box for providing further information about the service. For more information, see "Manage Services Dialog Box" on page 751.
Import	Opens the **Import Service** dialog box. For more information, see "Import Service Dialog Box" on page 755.
Delete	Removes the selected service from the list.
<services list>	A list of the available services: • **Service Name.** The native name of the service. • **WSDL Location.** The source of the WSDL.

Specify Traffic Information Screen

This wizard screen enables you to specify the capture file for the incoming or outgoing traffic.

To access	**Analyze Traffic** button, **Next**
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below:

UI Element	Description
	**Browse.** Allows you to select a capture file to import.
	Opens the **Traffic Filters** node in the Recording Options dialog box. This allows you to specify which IP addresses to include or exclude from the script. For details, see "Recording Options" on page 258.
	Opens the "SSL Configuration Dialog Box" below which allows you to add SSL certificates to analyze traffic from a secure server.
Capture file	The name of a capture file containing the server traffic, usually with a **cap** extension.
Incoming Traffic	The IP address and port of the server whose incoming traffic you want to examine.
Outgoing Traffic	The IP address of the server whose outgoing traffic you want to examine.
Record into action	The section into which you want to create the script: **vuser_init**, **Action**, or **vuser_end**. For actions you want to repeat, use the **Action** section. For initialization steps, use **vuser_init**.

SSL Configuration Dialog Box

This dialog box enables you to configure the SSL certificate for your traffic file.

To access	**Analyze Traffic** button, **Next,** 
Relevant tasks	"How to Add Content" on page 692

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Add Certificate.** Adds a new line to the certificate list.

	**Delete**. Removes the selected certificate.
<certificate list>	The properties of the certificate. Specify the following: • **IP.** The URL or IP address of the machine hosting the certificate file. • **Port.** Port of machine hosting the certificate file. • **File.** The path of the certificate file (with a **pem** extension) containing the private key. Use the **Browse** button to locate the file. • **Password.** A password for decrypting the certificate file.

Web Services - Preparing Scripts for Replay

Preparing for Replay Overview

After you create a script with Web Service calls, you prepare it for replay.

You can enhance it with custom error and log messages or with transactions. In addition, you can enhance your script with JMS functions, **jms_<suffix>** or XML functions, **lr_xml_<suffix>**. For more information, see the Function Reference (**Help > Function Reference**).

Run-time settings let you emulate real users more accurately, configure the run-time settings. These settings include general settings and Web Service specific settings. For details, see "Run-Time Settings" on page 332.

In certain cases, you may need to use the result of one Web Service call as input for another. To do this, you save the result to a parameter and reference it later. For more information, see "How to Prepare Scripts for Replay" on page 726.

Testing Web Service Transport Layers Overview

Web services can be sent over various transport layers. The transport layer is the protocol used to transport messages to and from the server.

VuGen allows you to configure the transport layer for your services. It fully supports HTTP/HTTPS and JMS (Java Message Service) transport layers.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see " User Handlers" on page 721.

Sending Messages over HTTP/HTTPS

HTTP is used for sending requests from a Web client, usually a browser, to a Web server. HTTP is also used to return the Web content from the server back to the client.

HTTPS handles secure communication between a client and server. Typically, it handles credit card transactions and other sensitive data.

The typical request and response mechanism is synchronous. In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls in conjunction with WS-Addressing. For details, see "WS-Addressing" on page 714.

JMS Transport Overview

JMS is a J2EE standard for sending messages, either text or Java objects, between Java clients.

There are two scenarios for communication:

Peer-to-Peer. Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.

Publish-Subscribe. Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

VuGen supports point-to-point communication by allowing you to send and receive JMS messages to and from a queue.

Before you can send messages over JMS transport, you need to configure several items that describe the transport:

- **JNDI initial context factory.** The class name of the factory class that creates an initial context which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- **JNDI provider.** The URL of the service provider which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- **JMS connection factory.** The JNDI name of the JMS connection factory.

In addition, you can set a timeout for received messages and the number of JMS connections per process.

You configure these settings through the JMS run-time settings. For details, see "JMS > Advanced Node" on page 374.

This section also includes:

JMS Script Functions

VuGen uses its API functions to implement the JMS transport. Each function begins with a **jms** prefix:

Function Name	Description
jms_publish_ message_topic	Publishes messages to a specific topic

jms_receive_ message_queue	Receives a message from a queue
jms_receive_ message_topic	Receives published messages to a specific topic on a subscription.
jms_send_ message_queue	Sends a message to a queue.
jms_send_receive_ message_queue	Sends a message to a specified queue and receives a message from a specified queue.
jms_subscribe_ topic	Creates a subscription for a topic.
jms_set_general_ property	Sets a general property in the user context.
jms_set_message_ property	Sets a JMS header or property for the next message to be sent, or uses a JMS header or property to filter received messages.

The JMS steps/functions are only available when manually creating scripts—you cannot record JMS messages sent between the client and server.

Unlike peer-to-peer communication that uses message queues, the publish-subscribe functions, **jms_publish_message_topic**, **jms_subscribe_topic**, and **jms_receive_message_topic**, are not supported for Web Service calls. To use these functions with Web Service calls, you must manually set up user handlers to generate the JMS message payload. For more information, see "How to Create a User Handler" on page 731.

For details about the JMS functions, see the Function Reference (**Help > Function Reference** or click **F1** on the function).

JMS Message Structure

Each JMS message is composed of:

- **Header.** contains standard attributes (Correlation ID, Priority, Expiration date).
- **Properties.** custom attributes.
- **Body.** text or binary information.

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

To override the default behavior, use a **jms_set_general_property** function before sending the message. Set the JMS_MESSAGE_TYPE property to TextMessage, BytesMessage, or Default. For Example:

jms_set_general_property("step1","**JMS_MESSAGE_TYPE**","BytesMessage");

For more information, see the Function Reference **(Help > Function Reference)**.

Asynchronous Messages Overview

You can use VuGen to emulate both synchronous and asynchronous messaging.

In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

This section also includes:

- "Sending Asynchronous Calls with HTTP/HTTPS" below
- "WS-Addressing" below

Sending Asynchronous Calls with HTTP/HTTPS

This following section describes how to use asynchronous calls in HTTP/HTTPS. You use a **Wait for Event** step to instruct Vusers to wait for the response of previous asynchronous requests before continuing. The listener blocks the execution of the service until the server responds.

When adding a Web Service Wait for Event step, you specify the following:

- **Quantifier.** The quantifier indicates whether the Vuser should wait for **ALL** events to receive a response or **ANY,** just one of them. **ANY** returns the name of the first event to receive a response. **ALL** returns one of the event names.
- **Timeout.** the timeout in milliseconds. If no events receive responses in the specified timeout, then **web_service_wait_for_event** returns a NULL.
- **Events.** a list all of the asynchronous events for which you want to wait.

When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

For task details, see "How to Send Messages over HTTP/S" on page 728.

When setting up an asynchronous message, you can set the location to which the service responds when it detects an event using WS-Addressing. For more information, see "WS-Addressing" below.

WS-Addressing

WS-Addressing is a specification that allows Web Services to communicate addressing information. It does this by identifying Web service endpoints in order to secure end-to-end endpoint identification in messages. This allows you to transmit messages through networks that have additional processing nodes such as endpoint managers, firewalls, and gateways. WS-Addressing supports Web Services messages traveling over both synchronous or asynchronous transports.

The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

An optional **WSAAction** argument allows you to define a SOAP action for instances where transport layers fails to send a message.

The following example illustrates a typical SOAP message using WS-Addressing, implemented in the background by VuGen.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
                xmlns:wsa="http://www.w3.org/2004/12/addressing">
    <S:Header>
        <wsa:MessageID>
            http://example.com/SomeUniqueMessageIdString
        </wsa:MessageID>
        <wsa:ReplyTo>

<wsa:Address>http://myClient.example/someClientUser</wsa:Address>
        </wsa:ReplyTo>
<wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
        </wsa:FaultTo>
        <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
        <wsa:Action>http://myserver.example/DoAction</wsa:Action>
    </S:Header>
    <S:Body>
         <!-- Body of SOAP request message -->
    </S:Body>
</S:Envelope>
```

In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
web_service_call("StepName=Add_101",
        "SOAPMethod=Calc.CalcSoap.Add",
        "ResponseParam=response",
        "AsyncEvent=Event_1",
"WSAReplyTo=212.199.95.138",
        "WSDL=http://lab1/WebServices/CalcWS/Calc.asmx?wsdl",
        "UseWSDLCopy=1",
        "Snapshot=t1153825715.inf",
        BEGIN_ARGUMENTS,
            "first=1",
            "second=2",
        END_ARGUMENTS,
        BEGIN_RESULT,
            "AddResult=Param_AddResult1",
        END_RESULT,
        LAST);
```

You can issue WS-Addressing calls in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, leave the **Async Event** box empty in the Transport Layer options. In Script view, remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

For task details, see "How to Send Messages over HTTP/S" on page 728.

Database Integration Overview

When testing your Web Service, it is vital that you use data that is accurate and up to date. If you use a snapshot of data from a past date, it may no longer be valid or relevant.

The database integration allows you to access values in a database during your test, ensuring that the data is up to date.

The database integration functions are useful in the following scenarios:

- "Connecting to a Database" below
- "Using Data Retrieved from SQL Queries" below
- "Validating Database Values after a Web Service Call" on page 718
- "Checking Returned Values Through a Database" on page 719
- "Performing Actions on Datasets" on page 721

Connecting to a Database

To connect to a database, you add a connection step to your script. A built-in Connection String Generator guides you in creating a database connection string specific to your database and credentials. You can also test your connection before inserting the step.

When running your script with iterations, virtual users only repeat the **Action** section of the script. If you include the database connection step in the **Action** section, the test will repeat it for each iteration. Virtual Users only repeat the **Action** section of the script, but not the **vuser_init** or **vuser_end** sections. Therefore, we recommend that you place the database connection step in the **vuser_init** section, and the disconnect step, **lr_db_disconnect** in the **vuser_end** section.

In cases where you only need to do one query and scroll through the data, you should also place the **Database: ExecuteSQL Query** step in the **vuser_init** section.

For task details, see "How to Send Messages over JMS" on page 727.

Using Data Retrieved from SQL Queries

In this scenario, the test fetches data from the database and uses it at a later point in the script, such as calls to the Web Service. Since the script retrieves the data during each test run, the data is up to date and relevant.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Web Service call	web_service_call with {<param_name>}

Disconnect from database	lr_db_disconnect

You can iterate through the results in two ways:

- save them to a simple parameter during each iteration
- use VuGen built-in iterations to scroll through the data

For more information, see the Function Reference (**Help > Function Reference**).

In the following example, the **vuser_init** section connects to the database and performs a database query.

```
vuser_init()
{
lr_db_connect("StepName=myStep",
    "ConnectionString=Initial Catalog=MyDB;Data Source=mylab.net;user
id =sa ;password = 12345;" ,
    "ConnectionName=MyConnection",
    "ConnectionType=SQL",
    LAST);
lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses",
        "DatasetName=ds1",
        LAST);
    return 0;
}
```

At the end of your test, disconnect from the database in the **vuser_end** section.

```
vuser_end()
{
    lr_db_connect("StepName=myStep",
        "ConnectionString=Initial Catalog=MyDB;Data
Source=LAB1.devlab.net;user id =sa ;password = soarnd1314;" ,
        "ConnectionName=MyConnection",
        "ConnectionType=SQL",
        LAST);
    return 0;
}
```

In the Action section, you include the steps to repeat. Note the use of the **Row** argument. In the first call to the database, you specify the first row with **Row=next**. To retrieve another value in the same row, use **current**.

```
Action()
{
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=next",
        "OutParam=nameParam",
        LAST);
```

```
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=city",
        "Row=current",
        "OutParam=cityParam",
        LAST);
/* Use the values that you retrieved from the database in your Web
Service call */
    web_service_call( "StepName=EchoAddr_101",
        "SOAPMethod=SanityService|SanityServiceSoap|EchoAddr",
        "ResponseParam=response",
        "Service=SanityService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227168459.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>{nameParam}</name>"
                "<street></street>"
                "<city>{cityParam}</city>"
                "<state></state>"
                "<zip></zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
return 0;
}
```

Validating Database Values after a Web Service Call

In this scenario, the test executes a Web Service call that modifies a database on the backend. The goal of this scenario is to validate that the resulting values in the database are correct.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Check the data	lr_checkpoint
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

For more information, see the Function Reference (**Help > Function Reference**).

The following example illustrates this process of checking the data:

```
Action()
{
/* A Web Service call that modifies a database on the back end. */
    web_service_call( "StepName=addAddr_102",
        "SOAPMethod=Axis2AddrBookService|Axis2AddrBookPort|addAddr",
        "ResponseParam=response",
        "Service=Axis2AddrBookService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227169681.inf",
        BEGIN_ARGUMENTS,
        "xml:arg0="
            "<arg0>"
                "<name>{Customers}</name>"
                "<city>{City}</city>"
            "</arg0>",
        END_ARGUMENTS,
        LAST);
/* Query the database by the cusotmer name that was modified by the
Web Service*/
    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = '{Customers}'
",
        "DatasetName=ds1",
        LAST);
/* Get the values retrieved by the database query. */
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);
/* Compare the actual value with the expected value stored in the
database. */
    lr_checkpoint("StepName=validateCustomer",
        "ActualValue={Customers}",
        "ExpectedValue={CustomerName}",
        "Compare=Equals",
        "StopOnValidationError=false",
        LAST);
return 0;
}
```

Checking Returned Values Through a Database

In this scenario, the user executes a Web Service call which returns an XML response. The goal of this scenario is to validate the response of the Web Service call against expected values. The expected values are stored in a database. The script fetches the expected results from a database and then compares them with the actual response.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call with **Result=<result_param>**
Execute an SQL query	lr_db_executeSQLStatement
Retrieve the expected data	lr_db_getvalue to <param_name>
Validate the data	soa_xml_validate with an XPATH checkpoints.
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

You can use the XML validation tool to create a checkpoint for the response data. When creating the validation step, use the database parameter that you retrieved through **lr_db_getvalue**.

The following example illustrates a typical validation of data returned by a Web Service call. The Validation step compares the actual expected results:

```
Action()
{
    web_service_call( "StepName=GetAddr_102",
        "SOAPMethod=AddrBook|AddrBookSoapPort|GetAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227172583.inf",
        BEGIN_ARGUMENTS,
        "Name=abcde",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = 'abcde' ",
        "DatasetName=ds1",
        LAST);
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);
    soa_xml_validate ("StepName=XmlValidation_1146894916",
        "Snapshot=t623713af7a594db2b5fef43da68ad59d.inf",
        "XML={GetAddrAllArgsParam}",
        "StopOnValidationError=0",
        BEGIN_CHECKPOINTS,
            CHECKPOINT,"XPATH=/*[local-name(.)='GetAddr'][1]/*[local-
```

```
name(.)='Result'][1]/*[local-name(.)='name'][1]","Value_Equals=
{CustomerName}",
        END_CHECKPOINTS,
        LAST);
    return 0;
}
```

For more information, see the Function Reference (**Help > Function Reference**).

Performing Actions on Datasets

VuGen lets you perform actions on datasets returned by SQL queries.

The **lr_db_dataset_action** function performs the following actions on datasets:

- **Reset.** Set the cursor to the first record of the dataset.
- **Remove.** Releases the memory allocated for the dataset.
- **Print.** Prints the contents of the entire dataset to the Replay Log and other test report summaries.

> Note that when you retrieve binary data through **lr_db_getvalue**, you cannot print its contents using the **Print** action.

For information about the syntax and usage of this function, see the Function Reference (**Help > Function Reference**).

Customizing Overview

VuGen provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are user handlers and configuration files.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see below.

Configuration files let you customize advanced settings such as security information and the WSE configuration.

- " User Handlers" below
- " Custom Configuration Files" on page 723

User Handlers

User Handlers are open APIs through which you can perform the following operations:

- Get and set the request/response SOAP envelopes
- Override the transport layer
- Get and set the request/response content type

- Get and Set values for LoadRunner parameters
- Retrieve a configuration argument from the script
- Issue messages to the execution log
- Fail an execution

You can set up a user handler directly in a script, or implement it through a DLL. You can apply the handler locally or globally.

For task details, see "How to Create a User Handler" on page 731.

For sample user handlers, see " User Handler Examples" on next page.

Handler Function Definitions

For basic implementation of a user handler, you define a user handler function within your Vuser script with the following syntax:

```
int MyScriptFunction(const char* pArgs, int isRequest)
```

The **pArgs** argument contains the string that is specified in **UserHandlerArgs** argument of **web_service_call** function.

The **isRequest** argument indicates whether the function is being called during processing of a Request (1) or Response (0) SOAP envelope.

The content of SOAP envelope is passed to a parameter called **SoapEnvelopeParam** for both requests and responses. After the function processes the SOAP envelope, make sure to store it in the same parameter.

To call the handler function, use the function name as a value for the **UserHandlerFunction** argument in the relevant Web Service Call step. For more information, see the Function Reference (**Help > Function Reference**).

Event Handler Return Codes

VuGen recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_ SUCCEEDED_AND_ MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in **SoapEnvelopeParam**.

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the
server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
```

```
    if (isRequest == 1) {
        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");
        //Manipulate the string...
        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");
        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
    return 0;
```

Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration. These files let you control the behavior of the test during run time.

The standard .NET configuration file, **mmdrv.exe.config**, is located in the VuGen installation folder. Some applications have their own configuration file, **app.config**.

You can customize the test run further, by filtering out the input or output. In addition, you can configure security information. such as token information and whether or not to allow unsigned test certificates.

For task details, see "How to Customize Configuration Files" on page 734.

User Handler Examples

This section illustrates several common uses for user handlers.

.NET Filters

You can apply a .NET filter to your messages using the user handler mechanism.

If you are familiar with Microsoft's Web Service Enhancements (WSE) 2.0, you can create a .NET filter and register it for incoming or outgoing SOAP messages. A .NET filter is a class that is derived from Microsoft.Web.Services2.SoapInputFilter or Microsoft.Web.Services2.SoapOutputFilter. By overriding the **ProcessMessage** function of this class, you can examine and modify the envelope's body and header.

To define the filter globally for the entire script, add the following lines to the script's default.cfg file below.

```
[UserHandler]
Function=LrWsSoapFilterLoader
Args=<Filters InputFilterClass="class name" InputFilterLib="lib name"
OutputFilterClass="class name" OutputFilterLib="lib name" />
Order=BeforeSecurity/AfterSecurity/AfterAttachments
```

The **InputFilterClass** parameter indicates the name of your class, and **InputFilterLib** indicates the name of the assembly in which the class resides. For example:

```
    web_service_call(
        ...
        "UserHandlerName=LrWsSoapFilterLoader",
        "UserHandlerArgs=<Filters
InputFilterClass=\"MyFilterNamespace.MyFilterClassName\"
InputFilterLib=\"MyAssemblyName\" />",
        BEGIN_ARGUMENTS,
        ...
        END_ARGUMENTS,
        ...
        );
```

Use SoapOutputFilter to examine an outgoing **web_service_call** request, and SoapInputFilter to examine the response from the server. Use **InputFilterClass** and **InputFilterLib** if your filter is derived from SoapInputFilter, or **OutputFilterClass** and **OutputFilterLib** if your filter is derived from SoapOutputFilter.

To define the filter for a specific step, add the following arguments to the **web_service_call** function.

```
UserHandlerName= LrWsSoapFilterLoader
UserHandlerArgs=<Filters InputFilterClass=\"class name\"
InputFilterLib=\"lib name\" OutputFilterClass=\"class name\"
OutputFilterLib=\"lib name\" />
UserHandlerOrder=BeforeSecurity/AfterSecurity/AfterAttachments
```

Overriding the Transport Layer

The following example shows a user handler function overriding the transport layer. VuGen does not automatically send the SOAP request over HTTP transport—instead it follows the transport method indicated in the custom handler.

After you receive a response, set the response envelope with the command:

```
lr_save_string(someResponseEnvelopeStr, "SoapEnvelopeParam");
```

To apply an alternate transport layer, specify **ReplaceTransport** as a value for the **UserHandlerOrder** argument. Define the transport layer in the handler.

```
web_service_call(

...

"UserHandlerFunction=<Transport HandlerFunction>",

"UserHandlerArgs=<handler arguments>",

"UserHandlerOrder=ReplaceTransport"

...

LAST);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the **web_service_call**:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
"UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
   "<cls>"
    "<i>123456789</i>"
    "<s>abcde</s>"
   "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
   LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the actual path and content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the **web_service_call**:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same **web_service_call**, modify the Web Service call as shown below:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;
FileName=C:\\temp\\results.discomap",
```

How to Prepare Scripts for Replay

This task describes how to prepare the script for replay and run it.

Assign Input Parameter Values

First save the output result to a parameter, and then reference that parameter in a later Web Service call.

1. **Save the output parameter**.
   a. In the **Step Navigator**, double-click the Web Service call whose output you want to use, to view its properties.
   b. In the left pane, select the output argument whose value you want to save to a parameter.
   c. In the right pane, select **Save returned value in parameter**. Specify a name in the **Parameter** box.
2. **Use the saved parameter for input.**
   a. In the **Step Navigator**, double-click the Web Service call whose input parameters you want to set.
   b. In the left pane, select the input argument for which to use the saved parameter.
   c. In the right pane, select **Value**, and click on the abc icon. The Select or Create Parameter box opens.
   d. Select the saved output parameter from the **Parameter name** list.
   e. To specify an input parameter in Script view, select the value you want to replace and select **Use Existing Parameters** from the shortcut menu. Select one of the available parameters.

Set the Run-time Setting - Optional

Open the run-time settings (F4) to configure JMS and VM settings. Click the **JMS > Advanced** node. For user interface details, see the "JMS > Advanced Node" on page 374.

Configure XSDs With any type Elements - Optional

For Web Services that have an XSD schema with an **Any** type element, `<xsd:element name="<Any_element>" type="xsd:anyType" />`, check that the script conforms with the following model:

```
BEGIN_ARGUMENTS,
                          "xml:Any_element="
                               "<Any_element>"
                               "<string>the string to send</string>"
                               "</Any_element>",
END_ARGUMENTS,
```

The actual SOAP may differ slightly, but as long as your script conforms to the above model, it will run properly.

You can also send complex type elements for the **<any>** type. For example:

```
"xml:Any_element="
                                        "<Any_element>"
                                        "<myComplexTypeName>"
                                        "<property1>123</property1>"
                                        "<property2>456</property2>"
                                        "</myComplexTypeName>"
                                        "</Any_element>",
```

Run the Script

Click **Replay > Run**. Observe the output log for relevant messages.

Review the Test Results

The Test Results viewer opens automatically after the test run. An X indicates a failed step.

Expand the nodes to see detailed information about the SOAP response and checkpoints. For details, see "Viewing Test Results" on page 214.

How to Send Messages over JMS

This task describes how to send messages using the JMS transport method.

1. ### Open the step properties

   In the **Step Navigator**, select the step whose transport you want to set, and then select **Show Arguments** from the shortcut menu.

2. ### Select the JMS transport method

   Select the **Transport Layer Configuration** node and choose **JMS Transport**.

   For UI details, see "New Web Service Call Dialog Box" on page 699.

3. ### Set the run-time settings - optional

   Configure the run-time settings, For details, see "JMS > Advanced Node" on page 374.

4. ### Send synchronous JMS messages - optional

   Once you create a Web Service call and designate the transport method as JMS, VuGen sends the JMS messages in a synchronous manner. If desired, specify the queue information.

5. ### Send asynchronous JMS messages - optional

To implement asynchronous messages over JMS, you send the request or retrieve the response using JMS steps—not Web Service calls.

a. Click within the script at the desired location. Select **Insert > New Step** and expand the **JMS Functions** node.

b. Select a JMS function: **JMS Send Message Queue** sends a message to a queue. **JMS Receive Message Queue** receives a message from the queue.

c. Click **OK** to open the JMS function properties.

d. Specify a queue name and click **OK** to generate the JMS functions.

For additional information about these functions, see the Function Reference (**Help > Function Reference** or click **F1** on the function).

6. ### Send messages over JMS using SOAP messages - optional

To send messages over JMS, using the SOAP message and without a Web Service call:

a. Record SOAP messages using a standard Web protocol.

b. Click within the script at the desired location. Select **Insert > New Step** and expand the **JMS Functions** node.

c. Select a JMS function: **Send Message Queue** or **JMS Receive Message Queue**.

d. Click **OK** to open the JMS function properties.

e. Specify a queue name and click **OK** to generate the JMS functions.

For details, see the Function Reference (**Help > Function Reference** or click **F1** on the function).

How to Send Messages over HTTP/S

This task describes how to send messages using the HTTP transport method.

1. ### Open the step properties

In the **Step Navigator**, select the step whose transport you want to set, and then select **Show Arguments** from the shortcut menu.

2. ### Select the HTTP/S transport method

Select the **Transport Layer Configuration** node and choose **HTTP/S Transport**.

3. ### Send a HTTP synchronous message - optional

To send messages in synchronous mode over HTTP, create a standard Web Service call, and do not enable the **Async Support** option.

4. ### Send asynchronous HTTP messages - optional

a. Choose **HTTP/S Transport** and select the **Async Support** option.

b. Type an event name in the **Async Event** box.

c. Click **OK** to generate the Web Service call.

d. Add a **Wait for Event** step. Select **Insert > New Step** and choose **Web Service Wait for Event**.

e. Specify a step name, a quantifier, and a timeout. Click **Add** and insert the name of the event that you defined in the previous step.

In Script view, VuGen indicates asynchronous messaging with the added parameter, **AsyncEvent**.

```
web_service_call("StepName=EchoString_101",
        "SOAPMethod=EchoRpcEncoded.EchoSoap.EchoString",
        "ResponseParam=response1",
        "Service=ExtendedECHO_rpc_encoded",
        "AsyncEvent=Event_1",
        "Snapshot=t1157371707.inf",
        BEGIN_ARGUMENTS,
        "sec=7",
        "strString=mytext",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "EchoStringResult=first_call",
        END_RESULT,
        LAST);
```

The **AsyncEvent** flag instructs the Vuser to wait for the response of previous asynchronous service requests.

5. ### Send an asynchronous message using WS-Addressing - optional

a. Select the **Async Support** option and provide an event name in the **Async Event** box. This can be an arbitrary name.

b. Select **WSA Support**. In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. Autodetect is useful when running the same script on several different machines. The server will reply to the specified location when the event occurs.

c. Click **OK** to save the settings.

d. Instruct the Vuser to wait for an event. Select **Insert > New Step** and add a **Web Service Wait For Event** step after the Web Service call step.

e. Specify a step name, quantifier, and timeout. To add an event name, click **Add**. The Web Service will wait for the specified event before responding.

f. Use the **Edit**, **Move Up**, and **Move Down** buttons to manipulate the events.

How to Define a Testing Method

This task describes how to select a testing method.

1. ### Open the step properties

In the Step Navigator, right-click the step whose response you want to test, and select **Show Arguments**.

2. **Select an argument**

   Select the Output Argument node. For details, see "New Web Service Call Dialog Box" on page 699.

3. **Select a testing method and choose an expected response**

   - To perform negative testing only, select the **Negative Testing** check box and choose **SOAP Fault** as the **Expected Response**.
   - To accept any type of SOAP response, select the **Negative Testing** check box and choose **Any SOAP** as the **Expected Response**.
   - To perform positive testing only, clear the **Negative Testing** check box.

4. **Verify function in the script**

   In Script view, VuGen indicates the testing method with the **ExpectedResponse** argument. In the following example, the script performs negative testing, indicated by the **SoapFault** value:

```
web_service_call("StepName=AddAddr_101",
        "SOAPMethod=AddrBook|AddrBookSoapPort|AddAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapFault",
        "Snapshot=t1189409011.inf",
        BEGIN_ARGUMENTS,
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
```

5. **Evaluate the SOAP fault value**

   When you replay a script that results in a SOAP fault, VuGen saves the fault to a parameter called **response**. To check the returned value of the SOAP fault, evaluate the **response** output parameter using **lr_xml_find.**

   In the following example, **lr_xml_find** checks for a **VersionMismatch** SOAP fault and issues an output message.

```
lr_xml_find("XML={response}",
                "FastQuery=/Envelope/Body/Fault/faultString ",
                "Value=VersionMismatch",
                LAST);
if (soap_fault_cnt >0)
     lr_output_message{"A Version Mismatch SOAP Fault occurred")
```

   For more information about **lr_xml_find**, see the Function Reference (**Help > Function Reference**).

How to Add a Database Connection

This task describes how to add a database connection step through Tree view.

1. **Open Solution Explorer**

   Select **View > Solution Explorer**.

2. **Select a section**

   Select the desired section: **vuser_init** or **Action**. To avoid repeating the connection sequence in every iteration, place it in the **vuser_init** section.

3. **Insert a database connection step**

   Select **Design > Insert in Script > New Step**. Choose the **lr_db_connect** step. The Database Connection dialog box opens. Specify a **Step Name**, **Connection Name**, and **Data Provider**, OLEDB or SQL.

4. **Create a database connection string**

   a. Click **Connection String Generator** to generate a database connection string specific to your environment.

   b. Indicate the connection properties:

   - **Server Name**
   - **Database Name**
   - **Authentication** method: Windows Authentication or User/password.
   - **Username** and **Password**

   c. Click **Test Connection** to verify that the information you provided is correct.

   d. Select an **SQL Provider**, OLEDB or SQL, and click **Generate**.

5. **Verify function in the script**

   Check that an **lr_db_connect** function was written to the script.

How to Create a User Handler

This task describes how to write a user handler for your script.

1. Prerequisite - Create a Web Service call

   Import a WSDL file and create a standard Web Service Call. For details, see "Adding Web Service Script Content - Overview" on page 685.

2. Define a user handler function

   Define a user handler before the Web Service call:

```
int MyScriptFunction(const char* pArgs, int isRequest)
{
...
}
```

3. Call the user handler function

   Call the handler function by specifying the function name as a value for the **UserHandlerFunction** argument. in the Web Service Call.

```
web_service_call(

...

"UserHandlerFunction=MyScriptFunction",

"UserHandlerArgs=<handler arguments>",



LAST);
```

4. Evaluate the handler function

   Evaluate the handler's return code to determine if it succeeded. Use the return codes as described in " User Handlers" on page 721.

```
//This function processes the SOAP envelope before sending it to
the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {
        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");
        //Manipulate the string...
        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");
        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
```

5. Create a DLL file - optional

   To define a user handler through a DLL, locate the API header file, **LrWsHandlerAPI.h** in the product's **include** folder.

   You can use a sample Visual Studio project located in the samples/WebServices/SampleWsHandler folder as a template for creating a handler. The sample retrieves the request and response envelope and saves it to a parameter. To use this sample, open it in Visual Studio and modify it as required. If you do not need to save the request/response to a parameter, you can remove that section of the sample.

   After editing the sample, save it and compile the DLL. When you compile the project, Visual Studio places the **<user_handler_name>.DLL** file in the **bin** folder. If you compile the project from another location, or if you want to copy the DLL from one machine to another, make sure to place it in the **bin** folder.

6. Configure the user handler - optional

   Declare the DLL user handler globally or locally.

   To apply the user handler globally to all requests in the script, add the following section to the **default.cfg** file in the script's folder.

```
[UserHandler]
Function=<name>
Args=<arguments>
Order=<BeforeSecurity/AfterSecurity/AfterAttachments>
```

- **Name.** The name of the DLL.
- **Args.** A list of the configuration arguments for the handler. Use the **GetArguments** method to retrieve the arguments in your handler.
- **Order.** The order in which Vusers process the user handler in requests: **Before Security**, **After Security**, or **After Attachments**. You can also use this argument to override the transport layer, by entering the value **Replace Transport**.

  Note: Setting the **UserHandlerFunction** property of a **web_service_call** function, overrides the definitions in the .cfg file.

  By default, user handlers are processed before the security. For request messages, Vusers process the attachments handler after the security handler. For responses, Vusers process the handlers in a reversed order. In typical cases the order does not matter, so any value is acceptable.

  To override the Transport layer, specify **Order=Replace Transport** and specify the new transport handler. If you implement the transport handler as a separate DLL, the **HandleRequest** function is called, while the **HandleResponse** function is ignored.

  To use the handler locally, for a specific request, add the following arguments to the **web_service_call** function:

```
UserHandlerName=<name1>
UserHandlerArgs=<args1>

User-
Han-
dlerOrder=<BeforeSecurity/AfterSecurity/AfterAttachments/Replace
                                    Transport>
```

7. Copy the user handler to all required machines

   Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the product's **/bin** folder.

   If you copy the script to another machine, it retains the handler information, since it is defined in script's folder.

8. Implement the user handler - optional

   To implement a user handler, you use the entry functions **HandleRequest** or **HandleResponse**. Both functions have a single parameter, **context**, whose properties you can set in your handler. Use the Get functions to retrieve properties, and Set functions to pass information from the replay framework to the handlers or between the handlers.

   - **GetEnvelope.** Gets the envelope content. For example, example:
     ```
     const char * pEnvelope = context->GetEnvelope();
     ```
   - **GetEnvelopeLength.** Gets the envelope length

- **SetEnvelope.** Sets the envelope content and length. For example:
```
string str("MySoapEnvelope...");
context->SetEnvelope(str.c_str(), str.length());
```
- **SetContentType.** Sets a new value for HTTP header content type
- **LogMessage.** Issues a message to the replay log
- **GetArguments.** Gets the configuration arguments defined for the current handler in order to pass it to the DLL
- **GetProperty.** Gets a custom property value
- **SetProperty.** Sets a custom property value

For more information, see the comments in the **LrWsHandlerAPI.h** file located in the product's **include** folder.

How to Customize Configuration Files

The following steps describe how to modify configuration files. For details, see " Custom Configuration Files" on page 723.

Locate the configuration file

Determine the location of the configuration file. The standard .NET configuration file, **mmdrv.exe.config** is located in the product's **bin** folder. Some applications have their own file, **app.config**.

Save the application's configuration file

If your application has its own app.config file:

- To apply the configuration information globally to all scripts, save the **app.config** file as **mmdrv.exe.config** in the **bin** folder, overwriting the existing file.
- To apply the configuration information locally, specifically for this script, copy the **app.config** file to the script's folder. This overrides the **mmdrv.exe.config** file, and remains associated with this script even when you copy it to other machines.

Set the security - optional

By default, VuGen allows unsigned certificates to facilitate testing. To disallow unsigned certificates, modify the **allowTestRoot** flag in the <security> section to **false**.

```
<security>
   <x509 storeLocation="currentuser" alllowTestRoot="false"
```

Web Services Snapshots - Overview

Vuser scripts based on the Web Services protocol utilize VuGen's Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the standard Snapshot pane UI, see "Snapshot Pane" on page 90.

The Snapshot pane enables you to view snapshots of Web service calls. When you display the Snapshot pane for a Web Services script, the left side of the Snapshot pane displays a tree view of the snapshot data; the right side of the Snapshot pane displays a text view of the snapshot data.

The tree view on the left of the Snapshot pane is composed of a number of nodes. An icon to the left of each node indicates the type of the node:

- **Element:** Indicates that the node represents an element in the XML file.
- **Attribute:** Indicates that the node represents an attribute in the XML file.
- **Value:** Indicates that the node represents a value in the XML file.

In addition to the basic Snapshot pane functionality, the Snapshot pane for Web Services scripts includes additional functionality. The UI for this additional functionality is described below.

To access	Select **View > Snapshot**, or click the **Show Snapshot Pane** button on the VuGen toolbar.
Relevant tasks	"How to Prepare Scripts for Replay" on page 726

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Response	Displays a snapshot of the SOAP response returned by the server.
Request	Displays a snapshot of the SOAP request sent to the server by the Web Service call.
	Opens the XPath Search dialog box which enables you to perform an XPath search of the snapshot.
	Displays the previous or next results of the XPath search.
	Displays or hides the XML **attribute** nodes in the tree view of the snapshot.
	Displays or hides the XML **value** nodes in the tree view of the snapshot.

, continued

<shortcut menu>	• **Copy Selection.** Copies the text that is selected in the text view to the clipboard. • **Search Community.** Performs a community search using the text that is selected in the text view as the search string. For details about performing a community search, see "Editor - Overview" on page 51. • **Copy XPath.** In the tree view, copies the XPath of the selected node to the clipboard. In the text view, copies the XPath of the XML element in which the cursor is located to the clipboard. • **Copy full value.** In the tree view, copies the full XML code of the selected node to the clipboard. In the text view, copies the full XML code of the XML element in which the cursor is located. • **Insert XML Check.** Opens the Insert XML Check dialog box that enables you to insert an **XML Find** step into the Vuser script. This option is available in the Response view only. This option is available for **attribute** and **value** nodes only. • **Save value in parameter.** Opens the Save Value as Parameter dialog box that enables you to save the selected value to a simple parameter. This option is available in the Response view only. This option is available for attribute and value nodes only. • **Save XML in parameter.** Opens the Save Value as Parameter dialog box that enables you to save the selected value to an XML parameter. This option is available in the Response view only. • **Create Correlation.** Opens the Correlation tab in the Design Studio. The text selected in the Snapshot pane appears as a manual correlation entry in the Design Studio. For details, see "How To Manually Correlate Scripts" on page 138. This option is available in the Response view only. This option is available for attribute and value nodes in the tree view, and when text is selected in the text view. • **Create Correlation Rule.** Opens the Add as Rule dialog box that enables you to add the selected text as part of a correlation rule. For details, see "Correlation Tab [Design Studio] Overview" on page 134. This option is available in the Response view only. This option is available for attribute and value nodes in the tree view, and when text is selected in the text view.

Database Connection Dialog Box

This dialog box helps you create a string to connect to your database.

To access	Click **Connection String Generator** in the Database Connection dialog box.
Relevant tasks	"How to Send Messages over JMS" on page 727
See also	"Connection String Generator Dialog Box" below

User interface elements are described below:

UI Element	Description
Connection String Generator	Opens the Connection String Generator. For details, see "Connection String Generator Dialog Box" below.
Step Name	The name or IP address of the database server.
Connection String	The string by which to connect to the database. Use the **Connection String Generator**.
Data Provider	The SQL provider: **OLEDB** or **SQL**.

Connection String Generator Dialog Box

This dialog box helps you create a string to connect to your database.

To access	Click **Connection String Generator** in the Database Connection dialog box.
Relevant tasks	"How to Send Messages over JMS" on page 727
See also	"Database Connection Dialog Box" above

User interface elements are described below:

UI Element	Description
Test Connection...	Tests the connection to the database.
Generate	Generates the database connection string and writes it in the **Connection String** field in the Database Connection dialog box.
Server Name	The name or IP address of the database server.
DB Name	The name of the database.
Authentication	The authentication method for the database: **Windows Authentication** or **User/password**. • **User Name, Password**. The credentials for the database.

SQL Provider	The SQL provider: **OLEDB** or **SQL**.

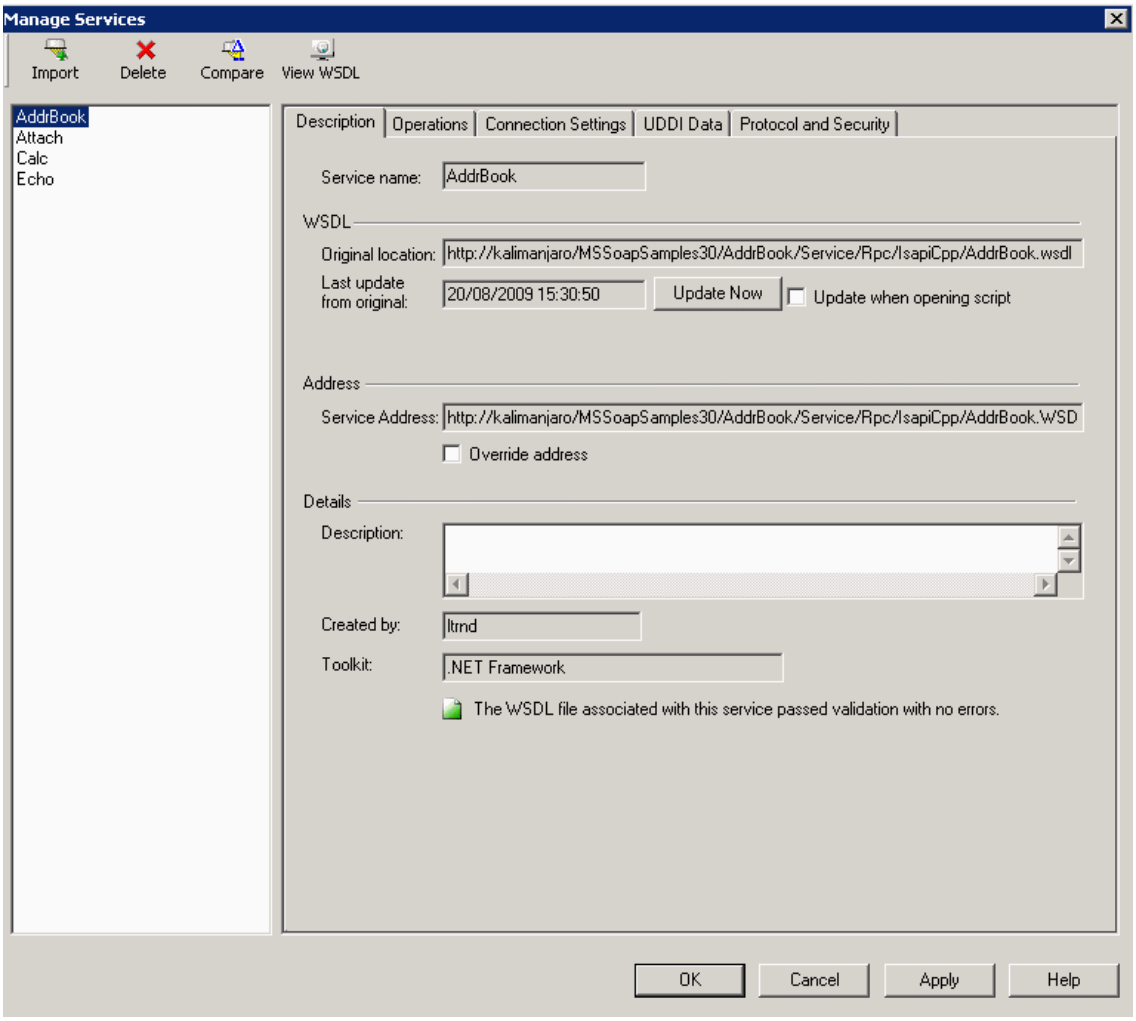
Web Services - Managing Services

Managing Services Overview

The Service Management window lets you manage a list of service entries for the current script. You can view and set the properties of each service entry.

You add service entries to the list by importing WSDL files. When you add a WSDL to the list, VuGen creates a working copy that it saves with the script—it is not global. Therefore, for each script that you create, you must import the desired WSDL files.

The Manage Services window provides quick access buttons for importing, deleting, and comparing services.



The tabs provide you with the ability to set the WSDL properties.

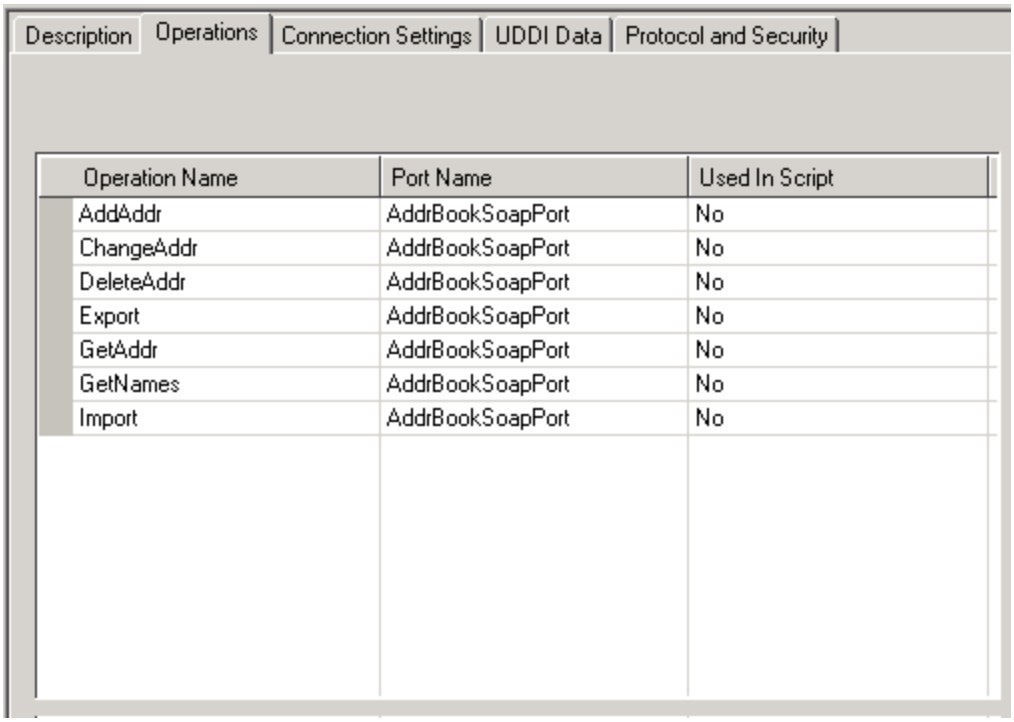
Description

The **Description** tab displays information about the service:

- **Original location.** The original source of the WSDL file (read-only).
- **Service name.** The name of the Web Service (read-only).
- **Last update from original.** The last date that the local copy was updated from the original source (read-only). You can update the version manually or retrieve it automatically each time you reopen the test.
- **Service address.** An endpoint address to which the request is sent. If required, you can override the endpoint specified in the WSDL file.
- **Created by**. The name of the user who originally imported the service (read-only).
- **Toolkit.** The toolkit associated with the script. You set this before importing the first WSDL file (read-only).

Operations

Each of the imported services may define multiple operations. The **Operations** tab indicates which operations are being used for the service selected in the left pane.



Connection Settings

In some cases WSDLs reside on secure sites requiring authentication. In certain instances, the WSDL is accessed through a proxy server.

VuGen supports the importing of WSDLs using security and WSDLs accessed through proxy servers. The following security and authentication methods are supported:

- SSL

- Basic and NTLM authentication
- Kerberos for the .NET toolkit

For more information about setting the connection information while importing the WSDL, see "How to Add and Manage Services" on page 746.

UDDI Data

You can view the details of the UDDI server for each service that you imported from a UDDI registry.

The read-only information indicates the URL of the UDDI server, the UDDI version, and the Service key.

Protocol and Security Settings

The Protocol and Security Settings tab shows the details of the security scenario applied to the script. If you did not choose a scenario, it uses the default **<no scenario>**. For more information, see "How to Add and Manage Services" on page 746.

This section also includes:

- "Importing Services" below
- "Comparison Reports" on next page

Importing Services

VuGen lets you import services for the purpose of creating a high-level tests with Web Service Call steps. Typically, you begin creating a script by importing a WSDL file.

The Import mechanism requires the following information:

- **Source.** The source of the WSDL: URL, File, UDDI, or Application Lifecycle Management. UDDI is a universal repository for services (Universal Description, Discovery, and Integration). Service brokers register and categorize published Web Services and provide search capabilities. The UDDI business registry is an example of a service broker for WSDL-described Web Services.
- **Location.** the path or URL of the WSDL, entered manually or by browsing.
- **Toolkit.** The toolkit to permanently associate with all services in the script for all subsequent imports and replays (only available for the first service added to the script). The toolkit setting instructs VuGen to send real client traffic using an actual toolkit—not an emulation.

  VuGen supports the .NET Framework with WSE 2 version SP3 and Axis/Java based Web Services Framework toolkits. VuGen imports, records, and replays the script using the actual .NET or Axis toolkit. By default, VuGen uses automatic detection to determine the most appropriate toolkit.
- **Connection Settings.** Authentication or proxy server information. This setting is useful for WSDLs residing on secure servers, or WSDLs that must be accessed via a proxy server.

If VuGen detects a problem with your WSDL when attempting to do an import, it issues an alert and prompts you to open the report. The report lists the errors and provides details about them.

For task details, see "How to Add and Manage Services" on page 746.

Comparison Reports

VuGen lists the differences between the files in a Comparison report.

You can configure the comparison settings, indicating which items to ignore during the comparison. For more information, see the "XML/WSDL Comparison Dialog Box" on page 756.

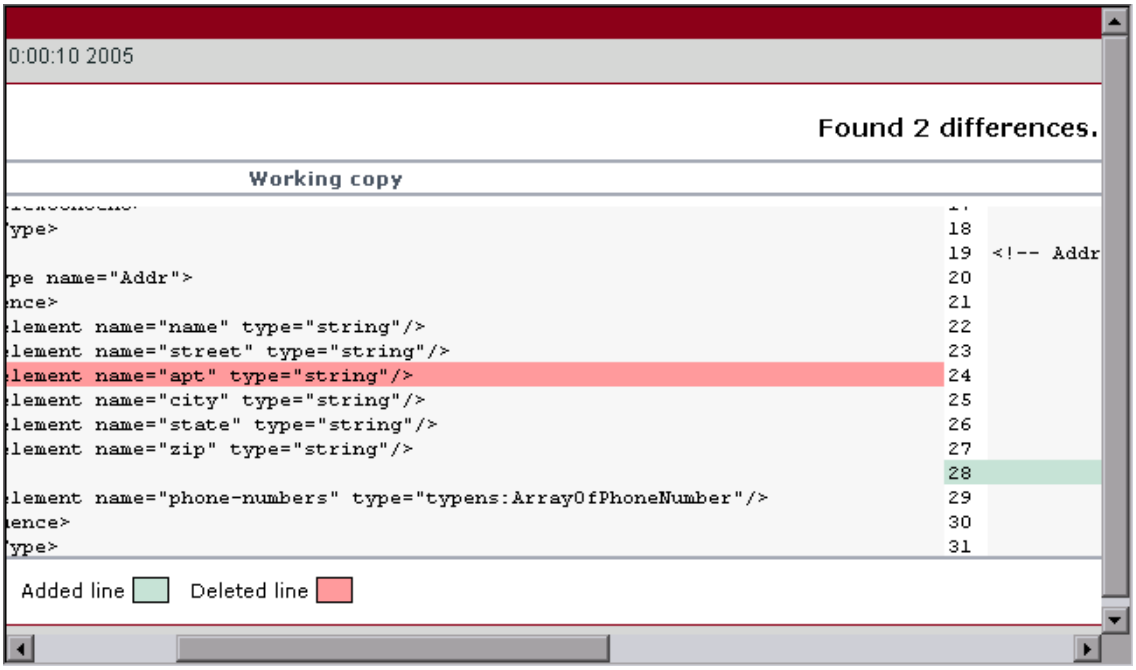
In WSDL Comparison reports, there are two columns— **Working Copy** and **Original File.** The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

In XML Comparison reports, each column displays the path of an XML file.

The Comparison report uses the following legend to mark the differences between the two files:

- **Yellow.** Changes to an existing element (shown in both versions).
- **Green.** A new element added (shown in the original file copy).
- **Pink.** A deleted element (shown in the working copy).

In the following example, line 24 was deleted from the original copy and line 28 was added.



Web Reference Analyzer

Many WSDL files reference other files such as XSD and other XML files. Before running a script, you may want to determine what these files are and if they are available.

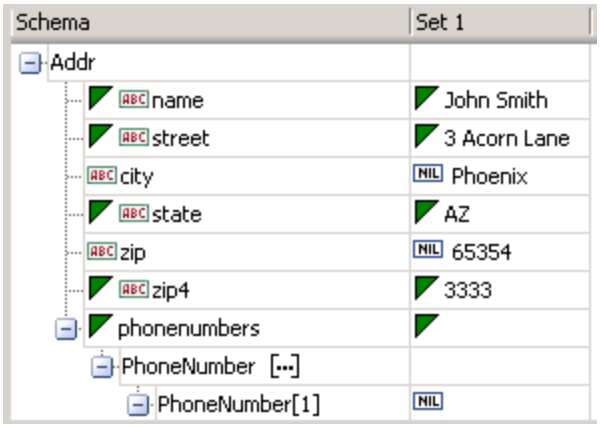
VuGen's WSDL Reference Analyzer checks the WSDL for dependencies, and lists them in the WSDL Reference Analyzer window and in a log file.

The Analyzer places the WSDL and its dependent files in a a zip archive file. It saves the dependency information to a log file, listing its path in the Analyzer window. For task details, see "How to Analyze WSDL Dependencies" on page 747.

XML Editing

The XML editor lets you view and edit XML and XSD schemas.

VuGen provides an editor that displays the XML structure according to a WSDL or XML schema. The grid-like display lets you view the XML in its hierarchy and set values for each of the elements. The left column represents the schema, while the other columns show the XML that is generated and its properties.



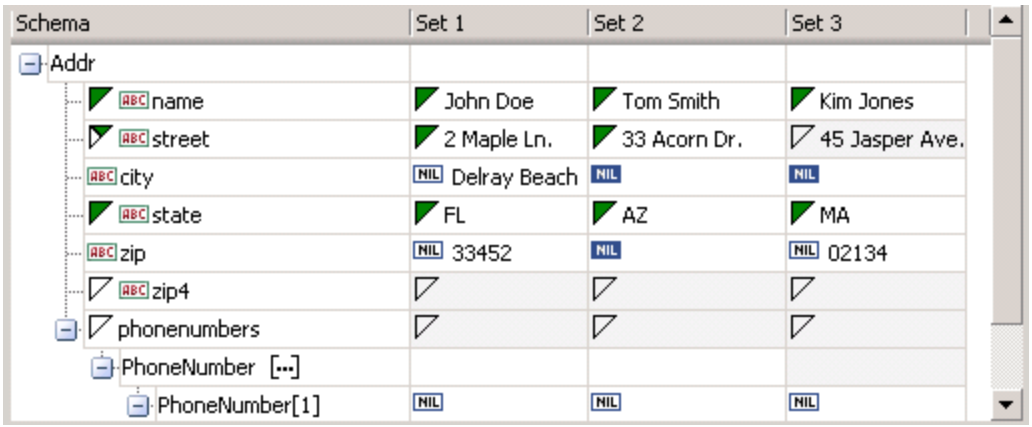
To set values for the elements, you can manually edit the XML or import XML files with the values.

The editor denotes optional elements with a small triangle in the upper left corner of the cell. A filled-in triangle indicates an included element. To exclude an optional element, click the small triangle to clear it.

When you exclude an element, the Editor works dynamically and removes the entire element from the XML code. When you re-include the element, the Editor adds it back into the XML.

Multiple Value Sets

Value sets are arrays that contain a set of values. You can create multiple value sets for your parameter and use them for different iterations or test runs.



You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

When using value sets, the number of array elements per parameter does not have to be constant. The exception to this is Choice, Derived, and <any> types, where the number of elements is fixed for all value sets.

For task details, see "How to Edit the Schema" on page 748.

XSD Schema Validation

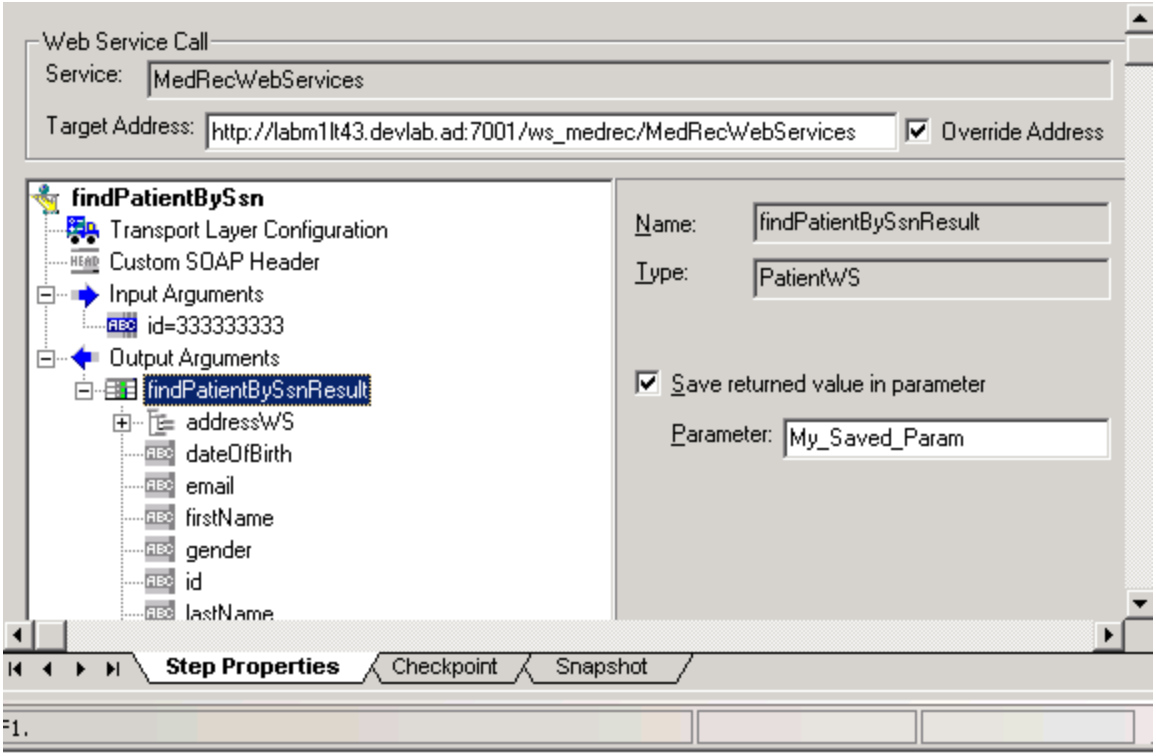
To perform complete XML validation, you select a parameter and indicate its corresponding XSD schema. When specifying a parameter for XML validation, you can use either an input or output parameter.

For input parameters, you use a parameter that you saved earlier. You can also select **Create/Select Parameter** from the **XML Source** dropdown list to create a new one. The parameter type should be XML parameter or a file type parameter where the file's content is XML.

Ideally, this parameter is XML with an ANY type element. By parsing the XSD, VuGen determines whether or not the WSDL is compliant with the schema.

You create output parameters in the Properties tab, by enabling **Save returned value in parameter** while selecting the output value. If you select the top node of the output parameter, VuGen saves the value as an XML parameter.

The following example shows an output array saved to a parameter, **My_Saved_Param**.



For information about defining parameters, see "Parameters" on page 227.

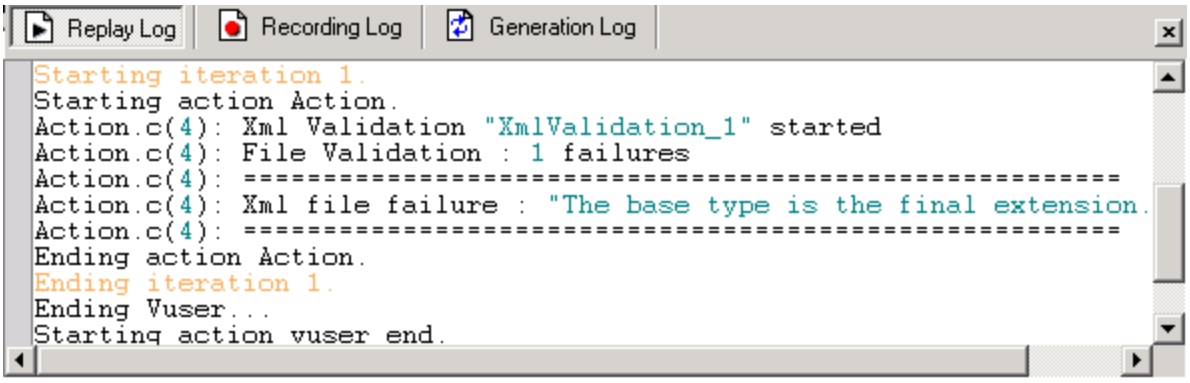
In the XML Validation dialog box, you provide the saved parameter as the **XML Source**.

Validation Results

By adding an XML Validation step to your script, VuGen performs the validations in runtime. You can determine the validation results in several ways:

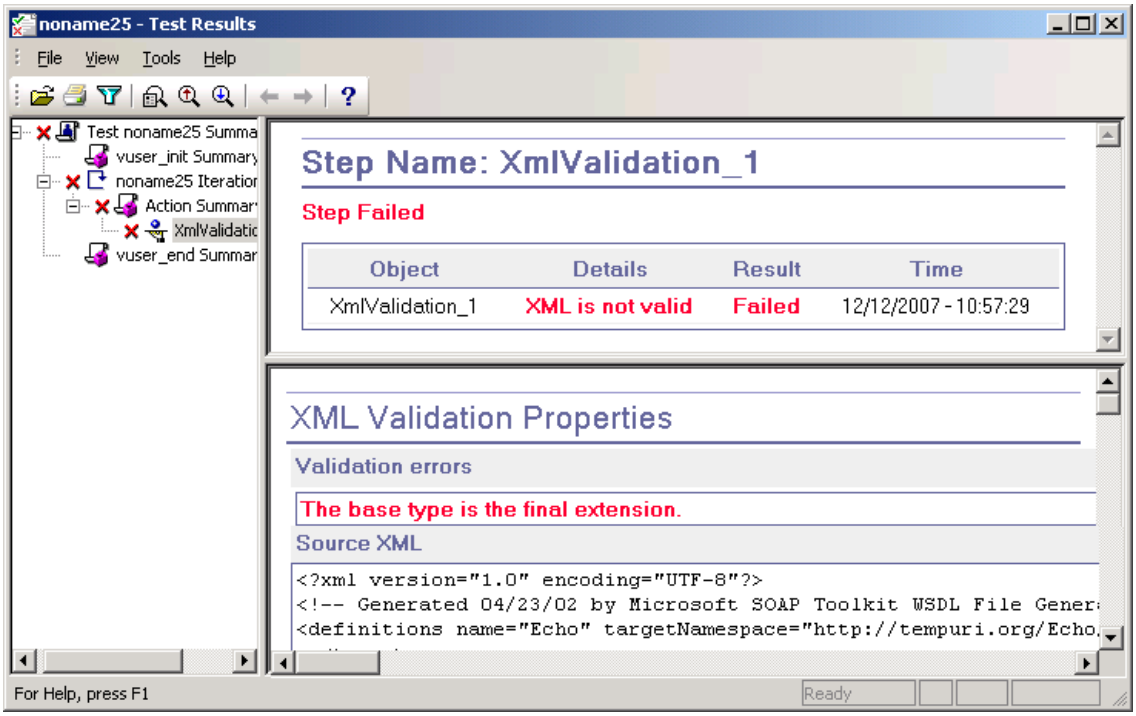
Replay Log

During replay, VuGen displays the results in the Replay log. We recommend that you enable the Extended Log run-time settings to obtain more details about the validation.



Test Results Report

The Test Results report shows the results of the validation step, along with the XML source code. To open the report, select **View > Test Results**.



Return Values

In addition to viewing the Replay log, you can evaluate the return values for **soa_xml_validate** to determine the outcome of the validation. The following table lists the possible return values:

Value	Description
0	Validation succeeded. XML is well-formed, compliant, and checkpoints are in order.
1	The parameter is empty or does not exist.
2	The XML is not well-formed.
3	The XML is well-formed, but not the XSD.
4	The XML does not comply with the XSD.
5	The values in the XML string do not match the values specified in the checkpoints' XPATH queries.
6	The XSD file could not be found.
7	An error, other than the ones listed in this table, occurred, such as an internal or unknown error.
8	The XSD file is not well-formed and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.
9	XML failed validation against the XSD file and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.

For more information about the function and its return values, see the Function Reference (**Help > Function Reference**).

REST Services and XML Validation

You can use the XML validation for non-SOAP Web Services, such as REST. The following example illustrates the collaboration between a REST Web Service and XML Validation.

```
// Save the content between the commas to a Design Run type parameter
web_reg_save_param("WCSParam_Text1",
        "LB=",
        "RB=",
        "RelFrameId=1",
        "Search=Body",
        "IgnoreRedirections=Yes",
        LAST);
//The service request submitted as a URL
        web_url("xml",
"URL=http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService=;

Oper-
ation=ItemSearch=;AWSAccessKeyId=123456789ABCD=;SearchIndex=Books=;
         Keywords=Cortisol=;Author=Tolstoy",
```

```
        "Resource=0",
        "RecContentType=text/xml",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
//Use the parameter's value in the validation statement.
        soa_xml_validate ("StepName=XmlValidation_1",
        "Snapshot=t3e9876543214.inf",
        "XML={WCSParam_Text1}",
        "StopOnValidationError=0",
        "XSD=C:\\Bugs\\67571_Rest\\AWSECommerceService.xsd",
         BEGIN_CHECKPOINTS,
    ...
        END_CHECKPOINTS,
        LAST);
```

To use a POST, PUT, or DELETE action, use **web_custom_request** as described in the LoadRunner Function Reference.

XML Queries

You can query an XML tree in order to locate and examine a specific element and value.

To search an XML file, you use a query in the standard XPATH syntax. To build a valid XPATH query, use the built-in Query Builder. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

How to Add and Manage Services

This task describes how to create a list of services that you can call from your test. Using the Manage Services window, you import services and configure their settings.

Open the Manage Services Dialog Box

Select **SOA Tools > Manage Services** or click the toolbar button to open the Manage Services dialog box.

Import a Service

Click **Import**. In the Import Service dialog box, select a WSDL source and browse to the location.

For **URL** type imports, the Browse button opens a new browser. Navigate to the WSDL and then close the browser. This action places the URL in the location box. For details, see the "Import Service Dialog Box" on page 755.

If your service requires authentication or uses a proxy, configure these settings before importing the WSDL. Expand the Import Services dialog box and click **Configure**. For details, see the "Connection Settings Dialog Box" on page 754.

Repeat this step for all the services you want to include in your test.

Get to Know the WSDL

Familiarize yourself with the WSDL. View its details as described in the "Manage Services Dialog Box" on page 751.

Click **View WSDL** to open the locally saved WSDL file in Internet Explorer and study its structure.

Check for WSDL Updates - Optional

Use the Comparison tool to check that the WSDL did not change since your last import or update.

First, set the comparison options. Click **SOA Tools > SOA Settings > XML/WSDL Comparison**. Specify what differences to ignore. For details, see "XML/WSDL Comparison Dialog Box" on page 756.

In the Manage Services window, click **Compare** to open a report comparing the working copy of the WSDL with the one at the original location.

If you discover changes in the Comparison report, click **Update Now** to retrieve the latest version of the WSDL from its source.

Override the Service Address- Optional

View the address in the **Service Address** box. This is the default endpoint address as retrieved from the WSDL. If you want to override it, select **Override address** and type in an alternate endpoint address for the service requests.

To return to the default address, clear the **Override address** option. For details, see the "Manage Services Dialog Box" on page 751.

Set a Security Scenario - Optional

Click the **Protocol and Security** tab to use WS-Security or another type of a security scenario.

For more information, see "Web Services - Security" on page 759.

How to Analyze WSDL Dependencies

This task describes how to use the Reference Analyzer to determine WSDL dependencies. For user interface details, see "WSDL Reference Analyzer Dialog Box" on page 757.

1. ### Open the Reference Analyzer

   Select **SOA Tools > WSDL Reference Analyzer**.

2. ### Select a source and target

   In the **Select WSDL file** box, indicate the location of the WSDL you want to analyze.

   In the **Output file path** box, indicate a location for the zip file.

3. ### Begin the analysis

   Click **Start Analyzing**. The Analyzer lists all of the dependencies in the output window along with their paths.

4. ### View the log

   View the results in the log window. To clear the results and perform another analysis, click

Clear Log.

How to Edit the Schema

This task describes how to use the XML editor to load values for the XML and XSD files. For user interface details, see "XML Editor Dialog Box" on page 757.

1. ### Open the XML editor

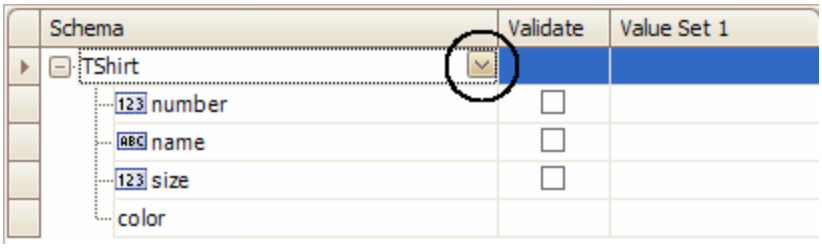
   Choose **SOA Tools > XML Editor**. The editor opens.

2. ### Load a schema

   Choose **Schema > Load** and select an XSD file for which you want to create value sets and edit.

3. ### Select a root

   If the schema has multiple root elements, select a root element. Click the small button adjacent to the root name to open the drop down box.

4. ### Assign values

   - Type entries into the **Values** column. To enter values manually, place the mouse over the element's icon to discover its data type. For more information about data types, see "How to Edit Values in the XML Grid" on next page.
   - To load values from an XML file, Click the **Load XML from file into the selected column** button.

5. ### Add more value sets - optional

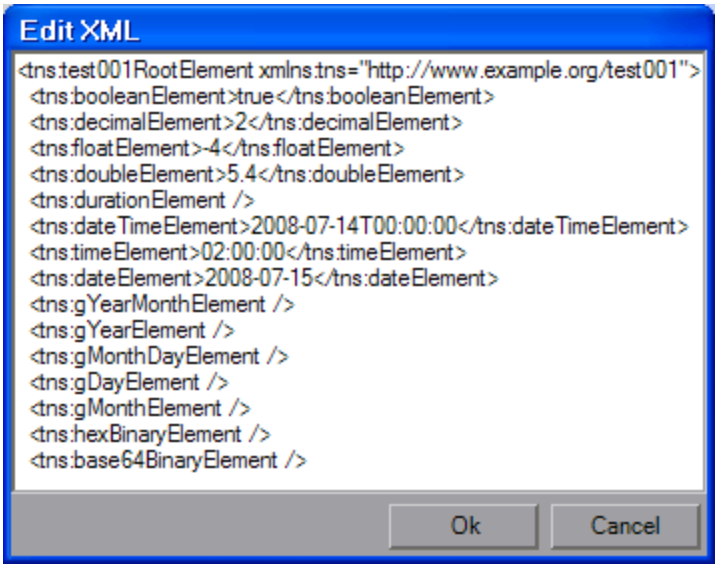
   Choose **Columns > Add** or click the **Add Column** button to add another value set. Set the values manually or by loading XML as described in the previous step.

   To add a column with identical values to the selected one, click the **Duplicate Column** button .

   To resize the columns of the value sets, click the divider in the column's title and drag your mouse to the desired width.

6. ### Edit the XML - optional

   To edit the actual schema, select one column and click the **Edit XML from the selected column** button. After the editing, click **OK**.



7. **Remove a value set - optional**

   To remove a column, select it and choose **Columns > Remove** or click the **Remove Column** button.

8. **Save a column and its values- optional**

   To save the XSD with the values of the current value set, select the column and choose **XML > Save** or click the **Save XML from the selected column** button.

How to Edit Values in the XML Grid

This task describes how to work in the XML grid.

Parameterization uses the Editor to display parameter elements and values as described in "Parameters" on page 227.

XML validation uses the grid to display the XSD schemas. Service Emulation uses the grid to display rules.

For user interface details, see "XML Editor Dialog Box" on page 757.

Enter the XML grid

Click within an XML grid. The XML grid is integrated into the XML Editor, Parameterization, XML Validation, and Service Emulation. Choose a root element if there are multiple ones.

Determine the data type

To determine an element's data type, place your mouse over the icon adjacent to the element name, such as **123**, **ABC**, **B64**, and so forth. The mouseover popup indicates the data type.

The grid recognizes element data types and provides an interface for setting the values. For example:

- For an **Int** type, the value cell contains a number scrolling control.

- For boolean, it contains a list box with the values **true**, **false**, **1**, or **0**.
- For a **Date** element, you can open a calendar to select a date.

Set the base64 properties

The schema indicates **Base64** elements with a **B64** button to the right of the value box. Click the button to open the Process Base 64 Data dialog box. For UI details, see "Process Base64 Data - Simple Data Dialog Box" on page 706.

Include or exclude optional elements

To include optional elements, fill in the green triangle adjacent to the element.

Include or exclude array elements

To include or exclude array elements, click on the green diamond in the square brackets.

- If the green diamond is visible, it includes the element.
- If the green diamond is removed, it excludes the element and all of its descendants.

The following example excludes an array element in several value sets.

Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [...]			
PhoneNumber[1]	[◆]	[◆]	[◆]
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	[◆]	[ ]	[◆]
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneNumber[3]	[◆]	[◆]	[ ]
description	Mobile	Mobile	Mobile
phone-number	555-5555	333-3333	123-4567

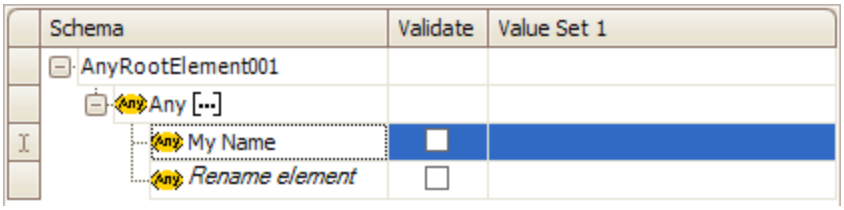
Add or remove array elements

To duplicate an array, right-click the parent node and select **Duplicate Array Element**.

To remove an array, right-click the parent node and select **Remove Array Element**.

Create a structure for <any> elements

- To add an array of **<any>** elements, select the parent **<any>** element and choose **Add Array Element** from the shortcut menu.
- To add a child **<any>** element, select the parent element and choose **Add child** from the shortcut menu.
- To add an additional child **<any>** element, select the child element and choose **Add sibling** from the shortcut menu.
- To provide a name for the **<any>** element, click the *Rename Element* text, and type in a name.



- To load values for an **<any>** element, choose **Load XML** from the shortcut menu.

How to Build an XML Query

This task describes how to use the Query Builder to locate and examine a specific element. For details, see "XML Queries" on page 746.

1. **Prerequisites**

   Create a script and run it at least once.

2. **Open the Find XML dialog box**

   In the Snapshot pane, Click **Find XPath**. In the Find XML dialog box, select **Request** or **Response.** For user interface details, see the "Find XML Dialog Box" on page 758.

3. **Specify a query**

   Manually type in an Xpath query or click **Query Builder**.

4. **Specify details for the Query Builder**

   If using the Query Builder, enter the search text in the appropriate boxes.

   - Enable the **Name** section to search for the name of a node or element.
   - Enable the **Namespace URI** section to search for a namespace.
   - Enable the **Text** section to search for the value of the element indicated in the Name box.
   - Enable the **Attributes** section to search for an attribute.
     - To add an attribute, click the **Add** button  . The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.

5. **Perform the query**

   Click **OK** in the XML Node Query dialog box. VuGen places the text of the query in the XPath query box. Click **Find Next** to begin the search.

Manage Services Dialog Box

This dialog box enables you to manage your WSDLs, provide authentication information, and set a security scenario.

To access	Use one of the following: • Click the **Manage Services** button  on the VuGen toolbar. • **SOA Tools > Manage Services**
Relevant tasks	"How to Add and Manage Services" on page 746

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Opens the Import Service dialog box.
	Removes the selected service from the list.
	Opens the WSDL Comparison Report showing the Working copy and Original copy of the WSDL side-by-side. To set the comparison settings, see "XML/WSDL Comparison Dialog Box" on page 756.
	Displays the WSDL in a browser.
<WSDL list>	A list of the imported WSDLs.
Description tab	Provides information about the WSDL, its endpoint address, toolkit, and update information.
Operations tab	Lists the operations of the service: **Operation Name**, **PortName**, and **Used in Script** (Yes or No). Click a column to sort the operations by that column's data. Click it again to reverse the sorting order.
Connection Settings tab	Allows you to provide authentication settings for the machine from which you are importing a service. For more information, see the "Manage Services Dialog Box" on previous page. **Note:** This only applies to URL and UDDI type imports.
UDDI Data tab	The UDDI server, UDDI version, and service key. For more information, see the "Manage Services Dialog Box" on previous page.
Protocol and Security tab	Allows you to view and set a security scenario for your Web Service calls. For more information, see below.

Description Tab

The following elements are displayed in the **Description** tab:

UI Element	Description
Update Now	Loads the latest version of the WSDL from its original location.
Created By	The name with which you logged in. You can edit this field and specify a different name. This is useful for sorting the services in reports (read-only).
Description	An editable field into which you can type information about the service.
Last update from original	The last date and time the WSDL was updated (read-only).
Original Location	The original location from where the WSDL was imported (read-only).
Override address	Enables you to enter an alternate endpoint for the service in the **Service Address** box.
Service Address	The endpoint of the service to which service requests are sent, retrieved from the WSDL file (read only). To override the default address, select **Override address**.
Service Name	The native service name in the WSDL file that is displayed by default when importing the service (read-only).
Toolkit	The toolkit associated with the service. You set this when you import the service (read-only).
Update when opening script	Updates the WSDL from its source each time you open the script.

Connection Settings Tab

The following elements are included:

UI Element	Description
Authentication	**Use Authentication Setting:** Enables you to enter credentials for authentication. • **Username, Password.** The user name and password to use for retrieving the WSDL. **Tip:** For users not in the default domain, type the domain name before the user name. For example, `domain1/alex_qc`.
Proxy	**Use Proxy Setting**. Enables you to enter proxy details and credentials. • **Server.** Name or IP address of proxy server. • **Port.** Port through which to access the WSDL. • **Username, Password.** the user name and password to be used for authentication. For users not in the default domain, type the domain name before the user name. For example, `domain1/alex_qc`.

UDDI Data Tab

The following elements are included:

UI Element	Description
Service Key	A unique identifier of the service on the UDDI server, used to locate the service definition when updating the service.
UDDI Server	The URL address and version of the UDDI server from which the service definition is imported.
UDDI Version	The version of the UDDI registry: 2 or 3.

Connection Settings Dialog Box

Enables you to provide authentication credentials and proxy server details for the machine hosting the WSDL file.

To access	For a new service: Select **SOA Tools > Import Service.** In the Import Services dialog box, click **Connection Settings**. For existing services: Select a service in the Service Management dialog box, and click the **Connection** tab.
Important information	Only available for services imported through a URL and UDDI.
Relevant tasks	"How to Add and Manage Services" on page 746

The following elements are included:

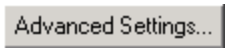
UI Element	Description
Authentication	**Use Authentication Setting:** Enables you to enter credentials for authentication. • **Username, Password.** the user name and password to use for retrieving the WSDL. **Tip:** For users not in the default domain, type the domain name before the user name. For example, `domain1/alex_qc`.
Proxy	**Use Proxy Setting**. Enables you to enter proxy details and credentials. • **Server.** Name or IP address of proxy server. • **Port.** Port through which to access the WSDL. • **Username, Password.** the user name and password to be used for authentication. For users not in the default domain, type the domain name before the user name. For example, `domain1/alex_qc`.

Import Service Dialog Box

Enables you to import WSDLs from a file system, a URL, Application Lifecycle Management, a UDDI, or Systinet.

To access	Use one of the following: • Select **Services > New > Import Services** • Select **New > Import Services** from the shortcut menu
Relevant tasks	"How to Add and Manage Services" on page 746

The following elements are included:

UI Element	Description
	**Browse.** Enables you to locate a service on the file system, through a browser, UDDI registry, or Application Lifecycle Management repository depending on your **Import WSDL from** selection.
	Opens the Connections Settings dialog box for configuring the authentication and proxy settings of the server hosting the WSDL. For details, see "Connection Settings Dialog Box" on previous page.
	Allows you to select a toolkit for the test. Choose **Automatic**, **.NET**, or **Axis**. The **Automatic** setting uses an algorithm to determine the most suitable toolkit.
	Opens the Application Lifecycle Management Connection dialog box to allow you to specify the ALM host machine.
	Begins the import process.
Select WSDL from	Location of WSDL. Browse for the information or enter it manually: • **URL:** Complete URL. Make sure to insert a complete URL—not a shortened version. • **File:** Full path and file name. • **UDDI:** UDDI registry ID. The Browse button opens the "Search for Service in UDDI Dialog Box" below.

Search for Service in UDDI Dialog Box

This dialog box enables you to locate a specific service from a UDDI registry.

To access	• In the Manage Services window, click **Import**. • In the Import dialog box, select **UDDI** in the **Select WSDL from** section. • Click  .
Relevant tasks	"How to Add and Manage Services" on page 746

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Begins the search for a service based on the text in the **All or part of the service name** box.
<service list>	An alphabetical list of all the services that match the string. The grid shows the following columns: **Service Name**, **Service Key**, **Service Description**, **Service WSDL**.
All or part of the service name	A string including the desired service name or part of the name. You do not need to use wildcard expressions. The following options narrow the search: • **Exact Match.** The service name must exactly match your text. • **Case Sensitive.** The case of service name must match the case of the specified text.
UDDI server inquiry address	The complete path for the inquiry on the UDDI server.
UDDI Version 2/3	The UDDI version of the services to display in the list.

XML/WSDL Comparison Dialog Box

This dialog box enables you to configure the settings for comparing different versions of a WSDL You can instruct the comparison tool to ignore specific differences such as case, comments, and so forth.

To access	**SOA Tools > SOA Settings > XML/WSDL Comparison.**
Relevant tasks	"How to Add and Manage Services" on page 746

User interface elements are described below:

UI Element	Description
Show only differences	Show only differences in the report—do not display the matching text.
Ignore case	Do not show case mismatches as differences.

, continued

Ignore comments	Do not mark mismatches in the comment as differences.
Ignore processing instructions	Do not mark mismatches in the processing instructions as differences.
Ignore namespaces	Do not mark mismatches in namespaces as differences.

WSDL Reference Analyzer Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	**SOA Tools > WSDL Reference Analyzer**
Relevant tasks	"How to Analyze WSDL Dependencies" on page 747

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Start Analyzing	Begins the analysis, showing all the results in the Log window.
Clear Log	Clears the log window and log file.
Open Containing Folder	Opens the folder containing the output file.
<log window>	A running log of the reference analysis.
Select WSDL file	The local path or URL of the WSDL file to analyze.
Output file path	A location for the output zip file.

XML Editor Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	**SOA Tools > XML Editor**
Relevant tasks	"How to Edit the Schema" on page 748

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Load Schema.** Opens an XSD schema file in the editor.
	**Add Column.** Adds a **Values** column to the right of the last column.

	**Duplicate Column.** Creates a new column with the same values as the selected column.
	**Delete Column.** Deletes the selected **Values** column.
	**Load XML.** Loads XML data from a file into the selected column.
	**Edit XML.** Opens the schema with the data of the selected column, in a text editor.
	**Save XML.** Saves the values from the selected column to an XML file.
<schema>	The XSD schema with the element values in grid format.
XML Values	An XML representation of the schema with the values of the selected column (right pane).

Find XML Dialog Box

This dialog box enables you to determine the dependencies of a WSDL file.

To access	**Find XPATH** in the **Snapshot** pane.
Relevant tasks	• "How to Build an XML Query" on page 751 • "How to Query an XML Tree" on page 522

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	Opens the XML Node Query dialog box for specifying query information.
XPath Query	The query expression. This can be entered manually or through the Query Builder.
Find	The SOAP message upon which to perform the search: **Request** or **Response**.

XML Node Query Dialog Box

User interface elements are described below:

UI Element	Description
	**Add Attribute.** Opens the Attribute Properties dialog box.
	**Delete.** Removes the selected attribute from the list.
	**Edit.** Enables you to edit the selected attribute.
Name	Searches for the name of a node or element.

Namespace URI	Searches for a namespace.
Text	Searches for the value of the element indicated in the **Name** box.
Attributes	Searches for an attribute in the attribute list.

Web Services - Security

Setting Security Overview

When building Web Service applications, there is a challenge in building scalable applications that are secure. You can secure Web Services by having the message sent over a secure transport, such as Secure Sockets Layer (SSL), but this is limited to point-to-point communication.

To allow you to send your messages securely, VuGen supports several security mechanisms, Security Tokens (WS-Security), and SAML.

For more information on tokens, see below. For more information on SAML, see "SAML Security Options" on page 762.

The following table lists the considerations for using each of the models.

Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service.
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework, such as Axis2 or Metro (WSIT).
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust.
You are having trouble using the new model or find the capabilities of the legacy more adequate for your needs	You are having trouble using the legacy model or you find the capabilities of the new model more adequate.

Note: If your WSDL is located in a secure location, you must provide the security information through the Manage Services dialog box. For more information, see the "Connection Settings Dialog Box" on page 754.

Security Tokens and Encryption

The WS-Security specification lets you place security credentials in the actual SOAP message. You accomplish this by instructing a client to obtain security credentials from a source that is trusted by both the sender and receiver. When a SOAP message sender sends a request, those security credentials, known as security **tokens**, are placed in the SOAP message. When the Web server receives the SOAP request, it does not need to send additional requests to verify the

integrity of the sender. The server verifies that the credentials are authentic before letting the Web Service execute the application. By not having to go back to the source of the credentials, this significantly improves the application's scalability.

To further secure Web Services, it is common to use digital signatures or encryption for the SOAP messages. Digitally signing a SOAP message verifies that the message has not been altered during transmission. Encrypting a SOAP message helps secure a Web Service by making it difficult for anyone other than the intended recipient to read the contents of the message.

The Web Services security mechanism associates security tokens with messages. This mechanism supports several security token formats to accommodate a variety of authentication requirements. For example, a client might need to provide a proof of identity or a security certificate.

To support WS-Security, VuGen allows you to create security tokens for your script. You can create multiple tokens and set their properties. After creating a token, you use it to sign or encrypt a SOAP message.

In certain instances, you do not send the token explicitly—you use the token for the purpose of signatures or encryption, without including the actual token in the SOAP envelope header. Using the **Add** option, you can indicate whether to send the actual token explicitly.

The available tokens are **Username and Password**, **X.509 Certificate**, **Kerberos Ticket**, **Kerberos2 Ticket**, **Security Context Token**, and **Derived Token**. The information you need to provide differs for each token.

- **User Name and Password.** The **User Name and Password** token contains user identification information for the purpose of authentication: **User Name** and **Password**.

You can also specify Password Options, indicating how to send the password to the server for authentication: **SendPlainText**, **SendNone**, or **SendHashed**.

- **X.509 Certificate.** This security token is a token based on an X.509 certificate. To obtain a certificate, you can either purchase it from a certificate authority, such as VeriSign, Inc. or set up your own certificate service to issue a certificate. Most Windows servers support the public key infrastructure (PKI) which enable you to create certificates. You can then have it signed by a certificate authority or use an unsigned certificate.

When you add an X.509 token to the Vuser script, you specify the **LogicalName**, **Store Name**, **Key identifier type**, **Key identifier value**, and **Store Location** arguments.

- **Kerberos Ticket/Kerberos2 Ticket.** (for Windows 2003 or XP SP1 and later) The Kerberos protocol is used to mutually authenticate users and services on an open and unsecured network. Using shared secret keys, it encrypts and signs user credentials. A third party, known as a KDC (Kerberos Key Distribution Center), authenticates the credentials. After authentication, the user may request a service ticket to access one or more services on the network. The ticket includes the encrypted, authenticated identity of the user. The tickets are obtained using the current user's credentials.

VuGen supports tokens based on both Kerberos and Kerberos2 security tokens. The primary difference between the Kerberos and Kerberos2 tokens is that Kerberos2 uses the Security Support Provider Interface (SSPI), so it does not require elevated privileges to impersonate the client's identity. In addition, the Kerberos2 security token can be used to secure SOAP messages sent to a Web Service running in a Web farm.

When you add a Kerberos token to the Vuser script, you specify a **LogicalName** for the token along with the **Host** and **Domain** names of the Web Services machine.

- **Security Context Token.** These tokens are security tokens that can be used repeatedly until they expire. SOAP message senders can use security context tokens to sign and/or encrypt a series of SOAP messages, known as a conversation, between a SOAP message sender and the target Web Service. The main benefits of this type of token are:
  - As long as the security context token has not expired, the SOAP message sender can use the same security context token to sign and/or encrypt the SOAP messages sent to the target Web Service.
  - Security context tokens are based on a symmetric key, making them more efficient at digitally signing or encrypting a SOAP message than an asymmetric key.
  - Security context tokens can be requested from one security token service by sending a SOAP message to another security token service.

When you add a **Security Context** token to the Vuser script, you specify values for the **LogicalName**, **Base Token**, **Issuer Token**, **End Point URI**, and **Add applies to** arguments.

- **Derived Token.** The Derived token is a token based on another existing token, excluding X.509 for which derivation is not supported. You need to specify a **LogicalName** and the **Derived From** token. If you remove the original token, then the derived token will no longer be available. Note that you cannot use a Derived type of token in a recursive manner.

For details about the token attribute in the script, see the Function Reference (**Help > Function Reference**).

Adding the Security Policy

To add a security policy to a section of your script, you enclose the relevant steps with **Web Service Set Security** and **Web Service Cancel Security** steps.

When you add a **Web Services Set Security** step to your script, VuGen adds a **web_service_set_security** function that contains arguments with the tokens, message signatures, and encryption that you defined.

```
web_service_set_security(
   SECURITY_TOKEN, "Type=USERNAME", "TokenName=mytoekn1",
   "UserName=bob", "Password=123", "PasswordOptions=SendNone",
   "Add=True", LAST);
```

Parameterization is not supported for the following arguments: **Token Type, Logical Name, Base Token, Issuer Token** or **Derive From** arguments.

Working with Message Signatures and Encrypted Data

When you add a security token to a SOAP message, it is added to the SOAP message in the form of an XML element in the WS-Security SOAP header.

The message, however, is exposed and therefore requires additional security. This is especially true when the credentials, including the password, are sent in plain text as it is with role-based security.

The two methods used to secure the data are digital signatures and encryption.

- **Digital Signatures.** Digital Signatures are used by message recipients to verify that messages were not altered since their signing. The digital signature is usually in the form of XML within the SOAP message. The recipient checks the signature to make sure it is valid. Certain

environments, such as WSE, automatically verify the signature on the SOAP recipient's computer.

- **Encryption.** Although the XML digital signature offers a mechanism for verifying that the message has not been altered since it was signed, it does not encrypt the SOAP message—the message is still plain text in XML format. To secure the message in order that it should not be exposed, you encrypt it, making it difficult for an intruder to view and obtain a user's password.

Parameterization is not supported for message signatures and encryption arguments. For details on adding message signatures and encryption to your script, see "How to Add Security to a Web Service Script" on page 776.

SAML Security Options

VuGen supports SAML (Security Assertion Markup Language) for Web Services. SAML is an XML standard for exchanging security-related information, called **assertions**, between business partners over the Internet. The assertions can include attribute statements, authentication, decision statements, and authorization decision statements.

SAML uses brokered authentication with a security token issued by STS (Security Token Service). The STS is trusted by the client and the Web Service to provide interoperable security tokens. SAML tokens are important for Web Service security because they provide cross-platform interoperability and a means of exchanging information between clients and services that do not reside within a single security domain.

You can set the SAML settings for an entire script or part of the script. For details, see "How to Add SAML Security" on page 777.

Note: You cannot apply SAML security and the standard Web Service (a **Web Service Set Security** step) security to the same step. To cancel Web Service security, insert a **Web Service Cancel Security** step.

Signing SAML Assertions

VuGen provides a method for signing an unsigned SAML assertion. As input, you provide the unsigned assertion, a certificate file, and the optional password. As output, VuGen provides the signed SAML assertion. For task details, see "How to Add SAML Security" on page 777.

Policy Files

SAML policy files follow the WSE 3.0 standard and define the attribute values for the SAML security. By default, VuGen uses the **samlPolicy.config** file located in the installation's **dat** folder.

When entering SAML security information, you can enter it manually in the properties dialog box, or you can refer to a policy file containing all of the security information. You can create your own policy file based on samlPolicy.config.

You can modify the policy file to include values for the security parameters, such as username and certificate information. When adding a SAML security step to your script, if you explicitly specify values for the security arguments, they override the values in the policy file.

If you make changes to the default policy file, we recommend that you copy the new policy file to your script's folder. Make sure to save custom policy files with a **.config** extension to insure that

they remain with the script, even when running it on other machines or calling it from the LoadRunner Controller.

To learn more about the SAML policy files, see the SAML STS example on the MSDN Web site. If you want to emulate SAML Federation behavior, copy the **samlFederationPolicy.config** file from the data folder to your script's folder, and specify it as the policy file.

Security Scenarios Overview

VuGen allows you to test Web Services that utilize advanced security and WS-Specifications. Such services can be written in various platforms such as WCF (Windows Communication Foundation), Metro (WSIT), and Axis2. For WCF services, VuGen also supports proprietary standards and transports.

You enable this support by setting up a security scenario. Each scenario represents a typical environment used in conjunction with Web Service calls. VuGen provides several built-in security scenarios that are commonly used. It applies the scenario's settings individually to each service.

For the built-in scenarios, the user interface lets you provide identity information where required. You can customize security, transport, proxy, and other advanced settings.

If you cannot find a scenario that corresponds to your environment, you can use the generic custom scenario.

For a "How To" guide on selecting a scenario, see Tips and Guidelines.

Choosing a Security Model

VuGen supports two models for configuring security for your Web Service calls: *Legacy* and *Scenario*. This chapter describes the Scenario security model. The Legacy model refers to the manual addition of Web Service Set Security steps, or the **web_service_set_security** function.

The following table lists the considerations for using each of the models.

Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework such as Axis2 or Metro (WSIT).
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust
You are having trouble using the new model or find the capabilities of the legacy functions adequate	You are having trouble using the legacy model or you find the capabilities of the new model more adequate

Private, Imported, and Shared Scenarios

To assign a security scenario to a specific service, use the Manage Services window. The **Protocol and Security** tab contains the interface to create and view security scenarios for individual services.

You can select a scenario in three ways:

- **Private scenario.** Create a new scenario by selecting one of the built-in ones and customizing it for your Web Service.
- **Imported scenario.** Use a scenario created at an earlier time. The scenario will be editable, and if someone modifies the original scenario, it will not affect you.
- **Shared scenario.** Load a security scenario already configured by another user from a remote location or the file system. You cannot edit this scenario's settings from the Manage Services window. If someone edits the scenario, it will affect your environment. You usually use this option after working with the product for some time and saving the scenario files.

Scenario Categories

The scenario describes the configuration of your Web Service. It contains information such as security, encoding, proxy, and so forth. VuGen provides a Security Scenario editor that allows you to configure the settings for each scenario.

To determine the scenario that best fits your service, refer to the sections below. If you are unsure which scenario to choose, we recommend to use the **Custom Binding** scenario. For more information, see " The Custom Binding Scenarios" on page 768.

Use the default **<no scenario>** for:

- simple Web Services where no advanced standards are required.
- scripts that use the legacy security model
- Web Services that require a specific security setting, not available in any of the existing scenarios.

If you select a built-in scenario and experience problems in replay, it is possible that no scenario was required and the problem is elsewhere. Reset the value to **<no scenario>.**

The built-in security scenarios are divided into the following categories:

Core Scenarios

The following table describes the built-in Core scenario.

Scenario Name	When to use
Plain SOAP	• Web services which do not require advanced standards • Web services which may require you to specify the WS-Addressing version

For this type of scenario, if your service uses WS-Addressing, specify the version.

Security Scenarios

The following table describes the built-in Security scenario.

Scenario Name	When to use
Username Authentication	• Client is authenticated with a username and password on the message level

For this type of scenario, specify the username/password, and if your service uses WS-Addressing, specify the version.

WCF Scenarios

The following table shows the scenarios for Web Services that utilize WCF. The WSHttpBinding-based scenarios are divided according to the way the client authenticates itself to the server. For example, if your client presents a user name and a password to the server, choose the **Username (message protection)** scenario. The user interface lets you provide the identity information in the form of a user name or a certificate as required.

WCF Scenario Name	When to use
WSHttpBinding - No Authentication	• Client uses the server's X.509 certificate for encryption • Client is not authenticated • Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - Windows authentication	• Client and server use Windows authentication • Security is based on Kerberos or SPNEGO negotiations • Communication may utilize advanced standards such as secure conversation and MTOM
wsHttpBInding - Certificate authentication	• Client uses the server's X.509 certificate for encryption • Client uses its own X.509 certificate for signature • Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - username (message protection) authentication	• Client uses the server's X.509 certificate for encryption • Client is authenticated with a username and password • Communication may utilize advanced standards such as secure conversation and MTOM

WSHttpBinding - username (transport protection) authentication	• SSL is enabled • Client is authenticated with a username and password • Communication may utilize advanced standards such as secure conversation and MTOM
WSFederationHttpBinding	• Client authenticates against the STS using a predefined scenario • Client uses the token given from the STS to authenticate against the server
Custom Binding	• Web Service that uses WS-* standards • WCF services of any configuration

Optimization Scenarios

The following table describes the built-in Optimization scenario.

Scenario Name	When to use
MTOM	• MTOM enabled Web services • Web Services which may require you to specify the WS-Addressing version

For MTOM type scenarios, if your service uses WS-Addressing, specify the version.

WCF Scenario Settings

This section describes the values required for the WCF security scenarios:

The WsHttpBinding Scenario

No Authentication (Anonymous)

In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication.

You specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For more information, see "Select Certificate Dialog Box" on page 785. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information.

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the

certificate was issued.

Windows Authentication

This WCF scenario uses Windows Authentication.

You declare the expected identity of the server in terms of its **SPN** or **UPN** identities. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.

Certificate Authentication

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For details, see "Select Certificate Dialog Box" on page 785. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username Authentication (Message Protection)

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

Specify only one of the following settings:

- **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.
- **Specify service certificate.** Browse for a service certificate. For details, see "Select Certificate Dialog Box" on page 785. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

- **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username (Transport Protection) Authentication

This WCF WSHttpBinding scenario enables SSL and authenticates the client with a user name and password on the message level.

Specify the following settings:

- **Username. Password.** The client's user name and password credentials.

The Federation Scenario

In the **WSFederationHttpBinding** scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server.

Therefore, two bindings are needed, one against the STS and another against the application server.

First, use the Security Scenario editor to define an STS binding. For more information, see " How to Create and Manage Security Scenarios" on page 778. When setting the binding against the application server, specify this file in the **Referenced file** box.

For the Federation scenario, specify the following server information:

- **Transport.** HTTP or HTTPS
- **Encoding.** Text or MTOM

For the Federation scenario, specify the following security information:

- **Authentication mode.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, IssuedTokenOverTransport, or SecureConversation
- **Bootstrap policy.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, or IssuedTokenOverTransport

For the Federation scenario, specify the following identity information:

- **Server certificate.** Browse for a server certificate. For more information, see the "Select Certificate Dialog Box" on page 785.
- **Expected server DNS.** the expected identity of the server in terms of its DNS. This can be **localhost** or an IP address or server name.

For the Federation scenario, specify the following STS (Security Token Service) information:

- **Issuer address.** The address of the issuer of the STS. This can be **localhost,** an IP address, or a server name.
- **Referenced binding.** The file that references the binding that contacts the STS (Security Token Service)

The Custom Binding Scenarios

The **Custom Binding** scenario enables the highest degree of customization. Since it is based upon WCF **customBinding**, it allows you to test most WCF services, along with services on other platforms such as Java that use
WS - <*spec_name*> specifications.

Use the **Custom Binding** scenario to configure a custom scenario that does not comply with any of the predefined security scenarios.

For the Custom Binding scenario, specify the following server information:

- **Transport.** HTTP, HTTPS, TCP, or NamedPipe
- **Encoding.** Text, MTOM, or WCF Binary

Specify the following security information:

- **Authentication mode.** None, AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SecureConversation, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Bootstrap policy.** For SecureConversation type authentication, specify a bootstrap policy: AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Net security.** the network security. Select None, Windows stream security, or SSL stream security. For services with HTTP transport, leave the default value, **None**. To enable SSL for HTTP, choose the HTTPS transport.

If your Web Service uses **Reliable messaging**, enable the option, and select **Ordered** or **Not Ordered**.

Identities

Your security settings may require you to provide identity details for either the client and server, or both of them.

An example of identity details for the client, are user name/password or an **X.509** certificate.

For identity information, provide one or more authentication details as required by the service:

Username, **Password**, **Server certificate**, **Client certificate,** or a custom Windows identity. For details about choosing a certificate, see "Select Certificate Dialog Box" on page 785.

Some scenarios require you to declare the expected identity of the server in terms of its DNS, SPN, or UPN identity.

- **DNS.** Provide the name of a server or use localhost.
- **SPN.** Provide the SPN identity in the `domain\machine` format.
- **UPN.** Provide the UPN identity in the `user@domain` format.

After setting the basic values, you can set advanced attributes as described in "Advanced Scenario Setting" below.

Advanced Scenario Setting

This section describes the Advanced scenario settings to customize a security scenario in the areas of Encoding, Advanced Standards, Security, or HTTP and Proxy.

Not all settings are relevant for all scenarios, so some of them might be disabled or hidden depending on the scenario.

Encoding

The Encoding tab lets you indicate the type of encoding to use for the messages: **Text**, **MTOM**, or **Binary**. The default is **Text** encoding.

For each of these encoding methods, you can choose a version of WS-Addressing:

- None
- WSA 1.0
- WSA 04/08

Advanced Standards

This tab lets you configure advanced WS- standards, such as Reliable Messaging and the Via address option.

If your service implements the **WS-ReliableMessaging** specification, enable the **Reliable Messaging** option and set the following options:

- **Reliable messaging ordered.** indicates whether the reliable session should be ordered
- **Reliable messaging version.** WSReliableMessagingFebruary2005 or WSReliableMessaging11

Via Address

In certain instances, you may need to send a message to an intermediate service that submits it to the actual server. This may also apply when you send the message to a debugging proxy. This corresponds to the WCF **clientVia** behavior.

In such cases it may be useful to separate the physical address to which the message is actually sent, from the logical address for which the message is intended. The logical address may be the physical address of the final server or any name. It appears in the SOAP message as follows:

<wsa:Action>http://myLogicalAddress<wsa:Action>

The logical address is retrieved from the user interface. By default, it is the address specified in the WSDL. You can override this address from the Manage Services dialog box.

Security

The Advanced security settings correspond to the **WS-Security** specifications.

For security scenarios that are based upon WCF WSHttpBinding, you can indicate the following settings:

- **Enable secure session.** Establish a security context using the WS-SecureConversation standard.
- **Negotiate service credentials.** Allow WCF proprietary negotiations to negotiate the service's security.

For **WSHttpBinding, Custom Binding**, or **WSFederationHttpBinding** WCF type scenarios, you can set the default algorithm suite and protection level:

Attribute	Meaning	Possible Values

Default Algorithm Suite	The algorithm to use for symmetric/ asymmetric encryption. These are the values from the SecurityAlgorithmSuite configuration in WCF:	• Basic128 • Basic128Rsa15 • Basic128Sha256 • Basic128Sha256Rsa15 • Basic192 • Basic192Rsa15 • Basic192Sha256 • Basic192Sha256Rsa15 • Basic256 • Basic256Rsa15 • Basic256Sha256 • Basic256Sha256Rsa15 • TripleDes • TripleDesRsa15 • TripleDesSha256 • TripleDesSha256Rsa15
Protection Level	Should the SOAP Body be encrypted/signed	None, Sign, and EncryptAndSign (default)

For **Custom Binding** or **WSFederationHttpBinding** WCF type scenarios, you can customize the security settings in greater detail. The following table describes the options and their values:

Attribute	Meaning	Possible Values
Message Protection Order	The order for signing and encrypting	• SignBeforeEncrypt • SignBeforeEncrypt-AndEncryptSignature • EncryptBeforeSign
Message Security Version	The WS-Security security version	A list of the current versions
Security Header Layout	The layout for the message header	• Strict • Lax • LaxTimeStampFirst • LaxTimeStampLast

Key Entropy Mode	The entropy mode for the security key.	• Client Entropy • Security Entropy • Combined Entropy

You can enable or disable the following options:

- **Require derived keys.** Indicates whether or not to require derived keys.
- **Require security context cancellation.** Disabling this option implies that stateful security tokens will be used in the **WS-SecureConversation** session (if enabled).
- **Include timestamp.** Includes a timestamp in the header.
- **Allow serialized token on reply.** Enables the reply to send a serialized token.
- **Require signature confirmation.** Instructs the server to send a signature confirmation in the response.

For X.509 certificates, you can specify values for the following items:

Attribute	Meaning	Possible Values
X509 Inclusion Mode	When to include the X509 certificate	• Always to Recipient • Never • Once • AlwaysToInitiator
X509 Reference Style	How to reference the certificate	• Internal • External
X509 require derived keys	Should X509 certificates require derived keys	• Enable - Yes • Disable - No
X509 key identifier clause type	The type of clause used to identify the X509 key.	• Any • Thumbprint • IssuerSerial • SubjectKeyIdentifier • RawDataKeyIdentifier

HTTP and Proxy

This tab lets you set the HTTP and Proxy information for your test.

HTTP(S) Transport

The following table describes the HTTP(S) Transport options:

Option	Meaning	Possible Values

Transfer mode	The transfer method for requests/responses	Buffered, Streamed, StreamedRequest, StreamedResponse
Max response size (KB)	The maximum size of the response before being concatenated	Default 65 KB
Allow cookies	Enable cookies	Enabled/Disabled
Keep-Alive Enabled	Enable keep-alive connections	Enabled/Disabled
Authentication scheme	HTTP authentication method	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous
Realm	The realm of the authentication scheme	Any URL
Require client certificate	For SSL transport, require a certificate	Enabled/Disabled

Proxy Information

If the Web service's transport uses a proxy server, you can specify its details in the **Security** tab. The following table describes the proxy options:

Option	Meaning	Possible Values
Use default web proxy	Use machine's default proxy settings	Enabled/Disabled
Bypass proxy on local	Ignore proxy when the service is on the local machine	Enabled/Disabled
Proxy address	the proxy server	Any URL
Proxy authentication scheme	HTTP authentication method on Proxy	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

WCF Extensibility

You can implement your own binding, behavior, or channel when using customBinding by defining the `assemblyPath` and `typeName` by modifying the configuration file `<script directory>/WSDL/@config/[your config].stss`.

The `assemblyPath` attribute should have a value of either the full path of the dll or its relative path to script directory.

The `typeName` attribute should have the full type name: `ns.typeName`.

Binding

Name the scenario attribute in the protocols element and provide the `assemblyPath` and `typeName` attributes.

The class you use for binding is inherited from `System.ServiceModel.Channels.Binding`.

Channel

Add a new element under the customization node. You can specify any name for the element, however the element must contain the two attributes:`assemblyPath` and`typeName`.

The class to use for binding is inherited from `System.ServiceModel.Channel.BindingElement`.

Note: This will work with customBinding scenarios only.

Behavior

Add a new element under the behaviors element (which is under `endpointBehavior`) and add the two attributes `assemblyPath` and `typeName`.

To bind the new element, implement the `System.ServiceModel.Description.IEndpointBehavior` class.

Note: If you inherit from `System.ServiceModel.Description.ClientCredentials`, the client credentials from this class will be used.

Examples of Channel and Behavoir

```
<protocols scenario="customBinding" uiType="customBinding"
xmlns="http://hp/ServiceTest/config">
<mode>Private</mode>
<customization>
<textMessageEncoding />
<preferlrhttpTransport />
<myChannel assemblyPath="CustomChannel.dll"
typeName="CustomChannel.WCFChannel" />
</customization>
<behaviors>
<endpointBehaviors>
<behavior>
<clientVia viaUri="qwqwq" />
<myBehavior assemblyPath="CustomBehavior.dll"
typeName="CustomBehavior.WCFbeahvior" />
</behavior>
```

```
</endpointBehaviors>
</behaviors>
</protocols>
```

An example of overriding the whole binding (the configuration may contain just one line):

```
<protocols scenario="userBinding" assemblyPath="WCFBinding.dll"
typeName=" WCFBinding.Binding"/>
```

Preparing Security Scenarios for Running

Parameterizing Security Elements

You can parameterize the security elements in a script independently. For example, in a username-based security scenario, you might want each Vuser or iteration to use a different user name.

Protecting Custom Headers

When an operation uses SOAP headers, VuGen does not automatically sign or encrypt them. To incorporate a protection scheme such as a signature or encryption, you must manually add the following information to the scenario's configuration file (.stss) in the **behavior** element:

- soapAction of the relevant operation
- The header XML name and namespace
- The protection level

The following example shows an outgoing message with the soapAction, **http://mySoapAction**. The XML element **header1** from namespace **http://myServiceNamespace** is encrypted and signed. The **header2** element from the same namespace is only signed.

```
<protocols ...>
              ...
              <behaviors>
                    <contractBehaviors>
                        <behavior>
                            <channelProtectionBehavior>
                                <protectionRequirements
action="http://mySoapAction">
                                    <incomingEncryptionParts>
                                        <header localName="header1"
namespace="http://myServiceNamespace" />

                                    </incomingEncryptionParts>

                                    <incomingSignatureParts>
                                        <header localName="header1"
```

```
namespace=" http://myServiceNamespace " />

                                        <header localName="header2"
namespace=" http://myServiceNamespace " />

                                    </incomingSignatureParts>

                                </protectionRequirements>

                            </channelProtectionBehavior>
                        </behavior>
                    </contractBehaviors>
                </behaviors>
            </protocols>
```

Emulating Users with Iterations

Many of the security scenarios establish a session with the server. For example, every scenario that uses **WS-SecureConversation** establishes a server session. This session is established when the first operation is executed and ends when the script is finished. By default, VuGen closes all sessions after each iteration and opens them again when the next iteration begins. This implies that every iteration simulates a new session and Vuser.

When working with multiple iterations, this may not be the desired effect—you may prefer to keep the original session active and not open a new session for each iteration. This applies when load testing through the LoadRunner Controller or when setting multiple iterations in the run-time settings.

You can override this behavior so that only the first iteration will establish a new session, while all subsequent ones will continue to use the open session. This simulates a user who repeatedly performs an action using the same session.

To determine which simulation mode to use, choose the one which is most appropriate to what you are simulating. For example, if you are simulating a load test where most of the actions are performed repeatedly by the same user in a single session, use the above configuration. If you are unsure, leave the default settings.

How to Add Security to a Web Service Script

This task describes how to add set the security for your Web Service calls. For details about Web Services security, see "Setting Security Overview" on page 759.

Insert a new Web Services Security Step

1. Place the cursor at the point at which you want to add the security settings. In most cases, it is best to place it in **vuser_init** so that the security scope will be applied to the whole script. To apply the security for specific calls, place it at the desired location.
2. Select **Insert > New Step** to open the Add Step dialog box.
3. Select **Web Service Set Security** and click **OK**. The Set Security Properties box opens.

Add a token - optional

1. Click **Add** to add a new token. The Add Token dialog box opens.
2. Select a token type. For details, see "Security Tokens and Encryption" on page 759.

   In the **LogicalName** box, assign an arbitrary name for the token to be used by VuGen in identifying the token.

   Add any relevant information, such as **User Name** and **Password** for the User Name and Password type token.

   To send the token explicitly in the SOAP envelope header, select **True**. To exclude the token from the SOAP envelope header, select **False**.

Add a message signature or encryption - optional

1. Click **Add > Message Signature** or **Add > Encrypted Data**.
2. Select a token to use with the message signature or encryption. Both signatures and encryptions require you to specify a token previously defined as the signing/encrypting token.
3. Specify a target token, or leave the field blank to apply the signature or encryption to the whole message body. For details, see "Security Tokens and Encryption" on page 759.

Set a message timeout - optional

To specify a time for which the message packet is considered valid, select **Time To Live** and specify a time in seconds.

Cancel the security settings - optional

To cancel the security settings at a specific point within the script, add a **Web Service Cancel Security** step at the desired point.

How to Add SAML Security

This task describes how to add SAML security for your Web Service calls. For more information about SAML security, see "SAML Security Options" on page 762.

For syntax information, see the Function Reference (**Help > Function Reference**).

1. ### Insert a new Web Services Security step
   a. Place the cursor at the point at which you want to add the security settings.
   b. Select **Insert > New Step** to open the Add Step dialog box.
   c. Select **Web Service Set Security SAML** and click **OK**. The properties box opens.
2. ### Insert a SAML assertion

   To add a SAML assertion method, add a **Web Service Sign SAML Assertion** step through the Add Step dialog box (**Insert > New Step**). Provide the unsigned assertion, a certificate file, and a password (optional).
3. ### Set the security policy - optional

   Specify a policy file, or leave it blank to use the default. If you manually enter values, they

override any values in the policy file. You must provide an Issuer URL, also known as the **STS URL**.

4. **Cancel the SAML settings - optional**

   To remove the settings at a specific point in the script, insert a **Web Service Cancel Security SAML** step.

How to Create and Manage Security Scenarios

The following steps describes how to create and customize a security scenario for a specific service.

1. **Open the Security Scenario Data dialog box**
   a. Click **Manage Services**. In the left pane, select the service for which you want to set the security scenario. If necessary, import a service, as described in "Import Service Dialog Box" on page 755.
   b. Select the **Protocol and Security** tab and click the **Edit Data** button. The Security Scenario Data dialog box opens.
2. **Create a scenario (if you do not have existing ones)**
   a. Choose **Private scenario** and select a built-on security scenario for the current service.
   b. In the **Scenario type** box, choose a scenario. For details, see "Choosing a Security Model" on page 763.
   c. Specify the required values for your scenario. For details, see "WCF Scenario Settings" on page 766.
   d. To specify a certificate (only applicable to some of the scenarios), click the Browse button adjacent to the **Client certificate** or **Specify service certificate** box to open the Select Certificate dialog box. For details, see the "Select Certificate Dialog Box" on page 785.
      - To retrieve a certificate from a file, choose **File** and locate its path.
      - To retrieve a certificate from a Windows store, Choose **Windows Store**. Select a Store location and name. Specify a search string—to search for all certificates, leave the **Search text** box empty. To search for a specific certificate, specify a substring of the certificate name. If required, specify a password for the private key. Click **Find** to generate the list of certificates found in the store.
3. **Load a security scenario (if you have existing ones)**
   a. To use an existing scenario with the ability to modify it, choose **Private scenario.** Click **Import**. In the Shared Scenario dialog box, select a stored scenario. If required, modify the settings as described in "WCF Scenario Settings" on page 766.
   b. To use an existing scenario without the option of changing it, choose **Shared Scenario**. Use the Browse button to open the Shared Scenario dialog box and select a stored scenario.

   Note: If someone modifies a shared scenario file at its source, it will affect your script.

4. **Configure advanced settings - optional**

   Click **Advanced** to configure the Proxy, Encoding, and other advanced setting. For most scenarios, the default settings are ideal. For details, see "Advanced Scenario Setting" on page 769. Click **OK** to save the security scenario.

5. **Modify an existing security scenario - optional**

   To create and modify security scenarios that will be available globally for all scripts—not just this specific service, use the Security Scenario editor. You can also use the editor to save the scenario so that others may load it.

   a. Choose **SOA Tools** > **Security Scenario Editor**.

   b. Click the **Load** button and browse for an existing **stss** scenario file.

   c. Modify the scenario settings as required

   d. Click **Save** or **Save as**.

6. **Protect SOAP headers - optional**

   Manually modify the **behavior** element in the scenario's configuration file

   a. In VuGen, open the Script view. Choose choose **View > Script View**.

   b. Click in the script editor and select **Open Script Directory** from the shortcut menu.

   c. Locate the security scenario's configuration file <*service_name*>**.stss** in **WSDL/@config** folder.

   d. Modify the behavior section of the file. For details, see "Protecting Custom Headers" on page 775.

7. **Set the iteration mode- optional**

   To configure your environment to use the same session for all iterations:

   a. Open the script root folder: In Script view, click inside the script and choose **Open Script Directory** from the shortcut menu.

   b. Open **default.cfg** file in a text editor.

   c. In the **[WebServices]** section, add in a row under the toolkit. If you are using the Axis toolkit or if you configured other settings, the file contents may differ.

   ```
   [WebServices]
   Toolkit=.Net
   SimulateNewUserInNewIteration=0
   ```

   d. Save and close the file.

   For details, see "Emulating Users with Iterations" on page 776.

How to Parameterize Security Elements

This task describes how to independently parameterize the security elements in a script.

1. **Open the Security Scenario Editor**

Select **SOA Tools > Security Scenario Editor**.

2. ### Set up a scenario for each Vuser

   Set up a scenario for each Vuser as described in " How to Create and Manage Security Scenarios" on page 778. We recommend you use the names **user1**, **user2**, and so forth, and save them in a new folder, **%script root%/WSDL/referencedConfig**.

3. ### Open the Parameter List window and create a parameter

   Select **Vuser > Parameters List**. Create a new parameter, **<ServiceName>_shared_config**. Replace the <ServiceName> with the case-sensitive name of the service you are testing. To determine the exact name of the service, click **Manage Services** to see the list of services.

4. ### Add parameter values

   In the values table, in each row add the file names of the security scenarios with their .stss extensions. You can use a relative path, relative to the script folder. Click **Add Row** to add multiple values. Close the Parameter List dialog box.

5. ### Call the parameter

   a. Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

   b. Select **Shared Scenario**. Click the Browse button and enter the parameter name, **<ServiceName>_shared_config**, in the test box.

Set Security Properties Dialog Box

This dialog box enables you to set the security properties for your Web Service calls.

To access	**VuGen > Insert > New Step > Web Service Set Security.** Click **OK**.
Relevant tasks	"How to Add Security to a Web Service Script" on page 776
See also	"How to Add SAML Security" on page 777
Important Information	If you have edited key algorithm or session algorithm values in the mmdrv.config file for an existing script, these values are replaced with the system default values.

WS -Security Tab

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<**Token Grid**>	Displays a unique number, type and name of all tokens that have been added.

Add Security Token

Enables you to select a token type:

Username and Password

UI Element	Description
Token Name	A meaningful name for the token.
Include nonce	If selected, an arbitrary number is used once to sign communication.
Username	Specify the username.
Password	Specify the password.
Password type	Specify the password as one of the following: • Text • Hash • None
Timestamp format	Specify the timestamp format: • Full • Created • None

X.509 Certificate Token

UI Element	Description
Token Name	A meaningful name for the token.
Certificate	If selected, an arbitrary number is used once to sign communication.
Reference Type	Specify the username.

Kerberos Token

UI Element	Description
Token Name	A meaningful name for the token.

UI Element	Description
Host	The host name of the server against which you want to authenticate. In most cases, it is the host portion of the service URL.
Domain	The Windows domain of the server against which you want to authenticate.

Kerberos2 Token

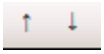
UI Element	Description
Token Name	Specify the name of the token.
Host	The host name of the server against which you want to authenticate. In most cases, it is the host portion of the service URL.
Domain	The Windows domain of the server against which you want to authenticate.

Add Message Signature

UI Element	Description
Signing token	The token to use for signing, usually an X.509 type. Select from the list of all added tokens.
Canonicalization algorithm	A URL for the algorithm to use for canonicalization. A drop down list provides common algorithms. If you are unsure which value to use, keep the default.
Transform Algorithm	A URL for the Transform algorithm to apply to the message signature. A drop down list provides common algorithms. If you are unsure which value to use, keep the default.
Inclusive namespace list	A list of comma-separated prefixes to be treated as inclusive (optional).
What to sign The SOAP elements to sign: SOAP Body, Timestamp, and WS-Addressing.	
Xpath (optional)	An XPath that specifies which parts in the message to sign. If left blank, the elements selected in the Signature options field are signed. For example, `//*[local-name(.)='Body'].`
Token (optional)	The target token you want to sign. Select from the drop down list of all added tokens. With most services, this field should be left empty.

<table>
<tr><td>  Add Message Encryption</td><td>
<table>
<tr><th>UI Element</th><th>Description</th></tr>
<tr><td>Encrypting Token</td><td>The token to use for encryption, usually an X.509 type. You can select from a list of all previously created tokens.</td></tr>
<tr><td>Encrypting Type</td><td>Indicates whether to encrypt the whole destination Element or only its Content.</td></tr>
<tr><td>Key algorithm</td><td>The algorithm to use for the encryption of the session key: RSA15 or RSAOAEP.

Note: If you have edited the mmdrv.config file with a custom key algorithm value for an existing script, this value is replaced with the system default value of RSA15.</td></tr>
<tr><td>Session algorithm</td><td>The algorithm to use for the encryption of the SOAP message. You can select from a list of common values.

Note: If you have edited the mmdrv.config file with a custom session algorithm value for an existing script, this value is replaced with the system default of AES128.</td></tr>
<tr><td colspan="2">What to encrypt</td></tr>
<tr><td>Xpath (optional)</td><td>An XPath that indicates the parts of the message to encrypt. If left blank, only the SOAP body is encrypted.</td></tr>
<tr><td>Token (optional)</td><td>The name of the encrypted token. A drop down box provides a list of all added tokens. With most services, this field should be left empty.</td></tr>
</table>
</td></tr>
<tr><td>  </td><td>Delete a token definition from the grid.</td></tr>
<tr><td>  </td><td>Up/Down. Positioning tools that allow you to set the
priority of the security elements.

Note: Make sure the security elements are positioned in order of their priority.</td></tr>
<tr><td>Exclude Timestamp</td><td>Removes the timestamp from the SOAP header before sending the security element to the server.</td></tr>
</table>

WS Addressing

The WS-Addressing tab indicates whether WS-Addressing is used by the service, and if so, its version number. You can also specify the IP address of the server to which you want the response to be sent.

Security Scenario Editor Dialog Box

This dialog box enables you to define security scenarios for your script.

To access	**SOA Tools > Security Scenario Editor**
Important information	You can also define scenarios for a specific service. For details, see " How to Create and Manage Security Scenarios" on page 778.
Relevant tasks	" How to Create and Manage Security Scenarios" on page 778

User interface elements are described below:

UI Element	Description
	New. Resets the editor for defining a new security scenario. If you made changes to the current scenario, it prompts you to save them.
	**Load.** Opens an existing shared scenario from a URL or file.
	**Save.** Saves the scenario file. If you have not saved the file at least once, it prompts you for a name.
	**Save as.** Saves the scenario file at a new location.
	**Help.** Opens the Online help for security scenarios.
	**Close.** Closes the dialog box.
	Opens the Advanced Setting dialog box for setting the encoding, reliable messaging, secure session information, and proxy configuration. For details, see "Advanced Scenario Setting" on page 769.
Scenario type	The security scenario type: No scenario or a sub-type of Core, Security, WCF, or Optimization scenarios.

Select Certificate Dialog Box

This dialog box enables you to search and locate a certificate from a file or Windows store.

To access	Select a scenario that uses a certificate in one of the following ways: • Open the Security Scenario Editor: Choose **SOA Tools > Security Scenario Editor**. • In the Manage Services dialog box, select the **Protocol and Security** tab and click the **Edit Data** button. Select a WCF scenario that uses a client or service certificate, such as **WsHttpBinding** or **Federation**. In the **Certificate** field, click the Browse button.
Important information	This only applies to security scenarios that allow you to specify client, server, or service certificates.
Relevant tasks	• " How to Create and Manage Security Scenarios" on page 778. • " How to Create and Manage Security Scenarios" on page 778.

Select Certificate from File

When you choose **File**, the dialog box shows the user interface elements described below:

UI Element	Description
...	**Browse.** Allows you to locate the certificate file with a .pem, .arm, .der, or .pfx extension.
File	The complete path of the certificate file.
Password (optional)	The password required to access the certificate.

Select Certificate from Windows Store

When you choose **Windows Store**, the dialog box shows the user interface elements described below (unlabeled elements are shown in angle brackets):

UI Element	Description
Find	Begins the search for the certificate.
Import from	The location of the certificate: • **Windows store** • **File**
Store location	The store location, for example **Current User**.
Store name	The store name, for example, **AuthRoot**.
Search text	The text to match in the certificate name.
Password (optional)	The password required to access the certificate.

<certificate list>	A list of the certificates in the Windows store sorted by Subject, Issuer, Private, Store Location, and Store Name.

Web Services Security Examples

This section illustrates several common security scenarios.

Authenticating with a Username Token

The following example illustrates the sending of a message level username/password token (a username token), where the user name is John and the password is 1234.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        LAST);
```

Signing a Specific Element with an X.509 Certificate

It is possible to sign only a specific element in a message. The following example signs a specific element using an XPATH expression:

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=myCert",
"StoreLocation=CurrentUser", "Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert", "TargetPath=//*[local-
name(.)='someElement' and namespace-uri(.)='http://myNamespace']",
        LAST);
```

Signing with an X.509 Certificate

The following example shows a script using an X.509 certificate for a digital signature.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=myCert",
"StoreLocation=CurrentUser", "Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        LAST);
```

Note: The certificate needs to be installed in the Windows certificate store. In the example above, you need to set the actual store name, store location, and subject name of your certificate.

Encrypting with a Certificate

The following sample encrypts a message with the service's X.509 certificate.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
```

```
"StoreLocation=CurrentUser", "Add=False",
        ENCRYPTED_DATA, "UseToken=serviceCert",
        LAST);
```

After you specify the details of your X.509 certificate, you can encrypt a specific XPATH in the message.

Since we want to generate a Subject Key Identifier, we set the Add value to **False**.

Authenticating with a Username Token and Encrypting with an X.509 Certificate

The following example sends a username token to the service and encrypts it with the server's X.509 certificate:

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
"StoreLocation=CurrentUser", "Add=True",
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myUser",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        ENCRYPTED_DATA, "UseToken=serviceCert", "TargetToken=myUser",
        LAST);
```

The **UseToken** and **TargetToken** properties indicate which token to use and which to encrypt. Their values reference the **LogicalName** property of the tokens.

Encrypting and Signing a Message

This example shows how to sign a message using a private key and then encrypt it using the service's public key.

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=myCert",
"StoreLocation=CurrentUser", "Add=True",
        SECURITY_TOKEN, "Type=X509","LogicalName=serverToken",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serverCert",
"StoreLocation=CurrentUser", "Add=False",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        ENCRYPTED_DATA, "UseToken=serverCert",
    LAST);
```

Referencing an X.509 Certificate Using a Hash

In certain cases, you may be unable to reference a certificate with a subject name. This example shows how to reference the certificate using its unique hash.

```
web_service_set_security(
   SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My",    "IDType=Base64KeyID",
"IDValue=pOl0+1iuotKLlO91nhjDg5reEw0=",
   "StoreLocation=CurrentUser", "Add=False",
```

```
  ENCRYPTED_DATA, "UseToken=serviceCert",
LAST);
```

Web Services - Service Emulation

Host Configuration Dialog Box

Enables you to view the host name and set the database connection settings.

To access	In the Host name toolbar, in the left pane, click the **Host Configuration** button  .
Important information	If your database requires a username and password, you need to enter them here.

User interface elements are described below:

UI Element	Description
Server	Server details: hostname\instance of the database.
Port	The port on the database server used by the emulated service.
Username\Password	Credentials for logging into the database server. **Default value: empty for localhost**

Host Name Pane

Displays the host name and lets you start or stop the server, and open the Host Configuration dialog box.

To access	Automatically displayed in the left pane.
Important information	The lower part of the left pane contains a toolbar for each of the available hosts.
See also	• "Host Configuration Dialog Box" above

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
	**Start Emulation Server.** Starts the Tomcat server and invokes the ServiceEmulationClient process.
	**Stop Emulation Server.** Stops the Tomcat server and the ServiceEmulationClient process.

	**Configure Host.** Opens the Host Configuration dialog box to set the database connection settings.
<hostname>	the machine hosting the displayed emulated services **Default value:** `localhost`

New Emulated Service Dialog Box

This dialog box lets you define an emulated service by indicating a WSDL and selecting a host.

To access	**Service > New Emulated Service**
Important information	Importing is not supported for WSDLs that import schemas. To import this type of WSDL, deploy it on a local server and then import it using the generated URL.

User interface elements are described below:

UI Element	Description
	**Browse.** Opens a browser (for **URL** selection) or the Open file dialog box (for **File** selection).
	Opens the Host Selection Dialog Box.
Select host	A dropdown list of the available host machines.
Select WSDL from	The source of the WSDL: • **URL** • **File**

Clear Service Call Log Dialog Box

This dialog box enables you to clear the service call log in the database. You can clear the whole log or a part of it, based on dates or labels.

To access	**Client Testing > Clear Service Log >**

User interface elements are described below:

UI Element	Description
From	Lower limit for the report generation through a time or label. For more information, see Labels. **Tip: Disable this option to clear all data from the beginning of the log until the *To* threshold.**

To	Upper limit for the report generation through a time or label. For more information, see Labels. **Tip: Disable this option to clear all data from the *From* threshold until the end of the log.**

New Label Dialog Box

This dialog box enables you to put a date and time stamp on incoming emulated service calls. These labels will allow you to filter you data when generating reports.

To access	**Client Testing > Add Label**
See also	• "Clear Service Call Log Dialog Box" on previous page

User interface elements are described below:

UI Element	Description
Current time	Time stamp associated with the label (read-only).
Label name	The label name as it will appear in the reports and filtering options.

Windows Sockets Protocol

Recording Windows Sockets - Overview

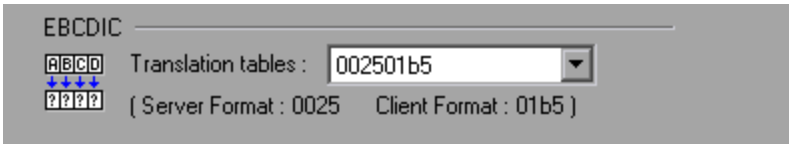
The Windows Sockets protocol supports applications which communicate over the TCP/IP protocol using a Microsoft WinSock DLL. The WinSock protocol allows you to see the actual data sent and received by the buffers.

The WinSock protocol records functions that relate to the sockets, data buffers, and the Windows Sockets environment. Using VuGen, you record your application's API calls to the Winsock.dll or Wsock32.dll. For example, you could create a script by recording the actions of a telnet application. After creating a Winsock Vuser script, you can view the recorded buffers as raw data or as a snapshot. For details, see "Windows Sockets Data" on next page or "Windows Sockets Snapshots - Overview" on page 793.

Translation Tables

You can display Windows Sockets data in EBCDIC format through a translation table.

A translation table allows you to specify the format for recording when using the WinSock single protocol, and for code generation when using a WinSock multi protocol. This applies to users running on mainframe machines or AS/400 servers. Both the server and client machines determine the format of the data from translation tables installed on your system. If your data is in ASCII format, it does not require translation.



The first four digits of the listbox item represent the server format. The last four digits represent the client format. In the above example, the selected translation table is `002501b5`. The server format is `0025` and the client format is `01b5` indicating a transfer from the server to the client. In a transmission from the client to the server, you would select the item that reverses the formats—`01b50025` indicating that the client's `01b5` format needs to be translated to the server's `0025` format.

The translation tables are located in the **ebcdic** folder under the VuGen's installation folder. If your system uses different translation tables, copy them to the **ebcdic** folder.

For details on selecting a translation table in the recording options, see the "WinSock Node" on page 331.

Windows Sockets Data

When you use VuGen to create a Windows Sockets Vuser script, your actions are recorded into the three sections of the script: **vuser_init**, **Action**, and **vuser_end**. In addition to the Vuser script, VuGen also creates data files:

- **snapshotdata.ws** contains the data that was transmitted or received during the recording session. VuGen's Snapshot pane displays the contents of the data file. Do not modify the contents of the **snapshotdata.ws** file.
- **data.ws** contains the data that is transmitted during the replay sessions, and is expected to be received. You can right-click any step in the Editor and then select **Show Arguments** to show the buffer content that is stored in **data.ws** for the selected step. Using the **Text View** tab of the dialog box that opens, you can edit the data that is stored for any data buffer.

Several LRS functions, such as **lrs_receive** and **lrs_send**, handle the actual data that is transferred between servers and clients. The data that is received or transmitted is stored in data buffers, which can be very large. In order to simplify the appearance of the Vuser script, the actual data is stored in external files—not in the C file. When a data transfer occurs, the data is copied from the external file into a temporary buffer.

The external file, **data.ws**, contains the contents of all the temporary buffers. The buffers' contents are stored as sequential records. The records are marked by identifiers indicating whether the data was sent or received, and the buffer descriptor. The LRS functions use the buffer descriptors to access the data.

The descriptors have one of the following formats:

```
recv buf index   number of bytes received
send buf index   number of bytes sent
```

The buffer index begins with 0 (zero), and all subsequent buffers are numbered sequentially (1,2, 3...) regardless of whether they are send or receive buffers.

In the following example, an **lrs_receive** function was recorded during a Vuser session:

```
lrs_receive("socket1", "buf4", LrsLastArg)
```

In this example, **lrs_receive** handled data that was received on socket1. The data was stored in the fifth receive record(buf4)—note that the index number is zero-based. The corresponding section of the **data.ws** file shows the buffer and its contents.

```
recv buf4 39
    "\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
    "\r\n"
    "\r\n"
    "SunOS UNIX (sunny)\r\n"
    "\r"
    "\x0"
    "\r\n"
    "\r"
    "\x0"
```

For task details, see "How to View and Modify Windows Sockets Buffers" on page 797.

Windows Sockets Snapshots - Overview

Vuser scripts based on the Windows Sockets Vuser protocol utilize VuGen's Snapshot pane.

- For an introduction to the Snapshot pane, see "Snapshot Pane - Overview" on page 54.
- For details on how to work with the Snapshot pane, see "How to Work with Snapshots" on page 57.
- For details on the Snapshot pane UI, see "Snapshot Pane" on page 90.

When you open a Windows Sockets Vuser script, VuGen's standard Snapshot pane functionality is available. For Windows Sockets Vuser scripts, the Snapshot pane displays snapshots of the recorded data buffers.

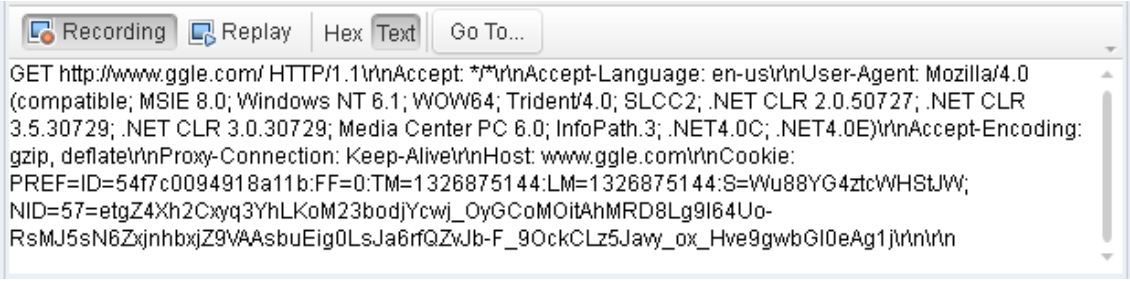
To display a specific buffer in the Snapshot pane:

- In the Editor, select the step that contains a reference to the required buffer.
- In the Step Navigator, double-click the step that contains a reference to the required buffer.

You can view the buffer snapshots in either **Text** view or **Hex** view.

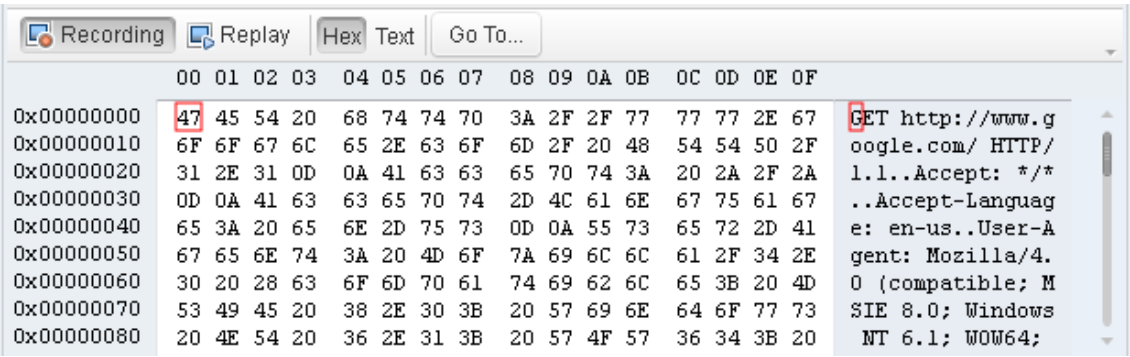
Note that VuGen stores the snapshot data as read-only data. You cannot modify the contents of the snapshots. However, you can modify the buffer data that is associated with any of the steps in a Vuser script. To modify the buffer data, right-click the required step in the Editor and then select Show Arguments. The Text View tab of the dialog box that opens lets you modify the buffer data. For details, see "How to View and Modify Windows Sockets Buffers" on page 797

The **Text** view shows the buffer data as text.



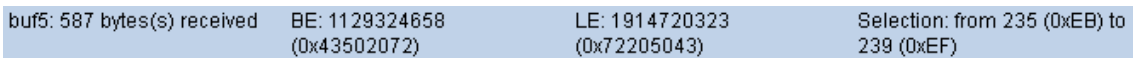
The **Hex** view shows the buffer data in hexadecimal representation. The data is displayed in three columns:

- The left column shows the offset of the first character in each row.
- The middle column shows the hexadecimal value of the data.
- The right column shows the data in ASCII format.



The status bar below the buffer snapshot displays information about the buffer and the data selected in the buffer:

- **Buffer number.** The buffer number of the displayed buffer.
- **Buffer size.** The total number of bytes in the buffer.
- **Buffer type.** The type of buffer—received or sent.
- **Data.** The value of the data that is selected in the buffer, in decimal and hexadecimal formats. Both big endian and little endian sequences are displayed.
- **Offset.** The offset of the selected data from the beginning of the buffer. If you select multiple bytes, the offset displays the range of the selection.



Navigating within the Buffer Data

- To go to a specific offset within the buffer (absolute), click **Go To**. In the Go To Offset dialog box enter an offset value, and then click **Apply**.
- To jump to a location relative to the selected entry, click **Go To**. In the Go To Offset dialog box, click **Advance by** , specify the number of bytes to advance, and then click **Apply**.

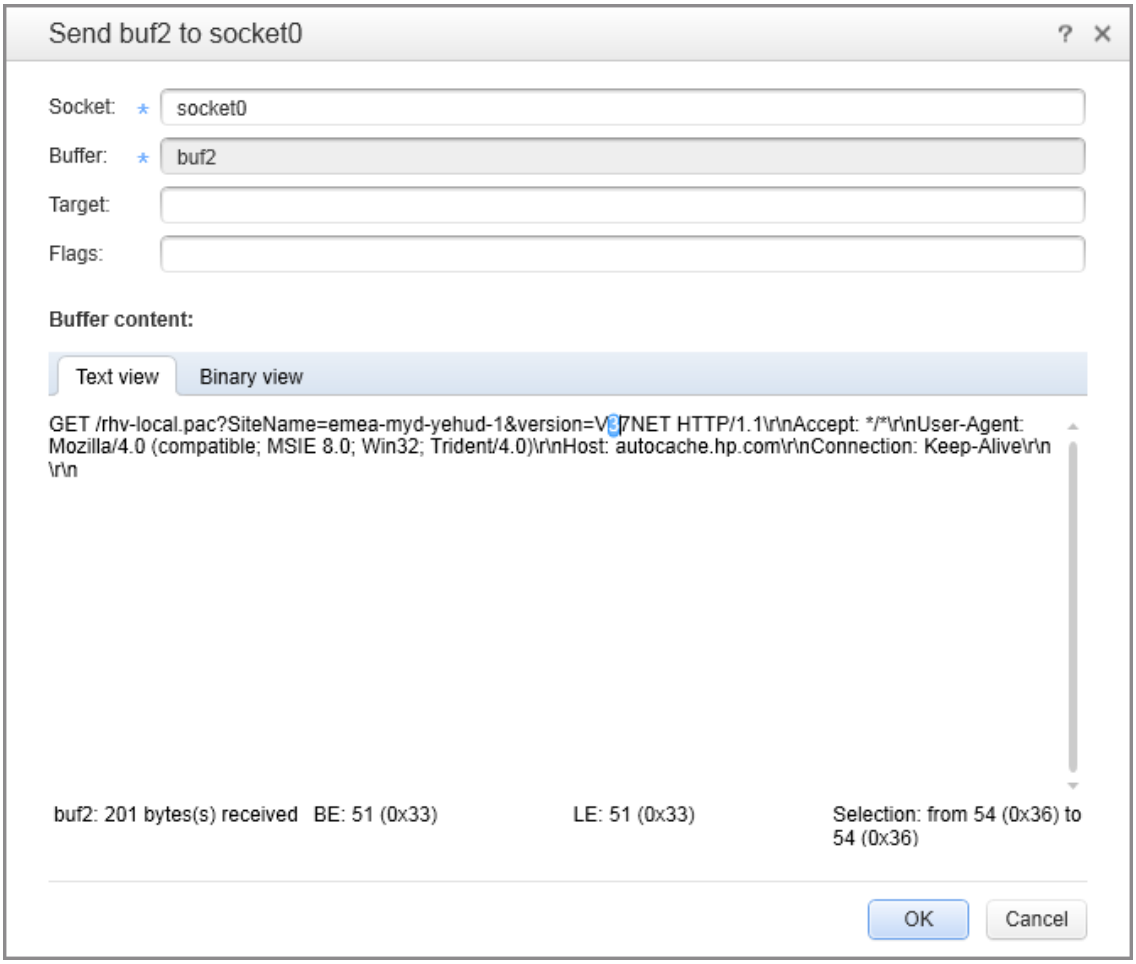
Data Navigation Tools

VuGen provides you with **Go To** functionality to help you to navigate through the data in the Snapshot pane. This helps you to identify and analyze a specific values in the snapshot. You can move around within the data buffer by specifying an offset. You can indicate the absolute location of the data, or a location relative to the current position of the cursor within the buffer. You can also select a range of data, by specifying the starting and end offsets. For details on the dialog box options, see "Go To Offset Dialog Box" on page 800.

Buffer Data Editing

You can perform all of the standard edit operations on buffer data: copy, paste, and delete. To perform edit operations on buffer data, right-click the required step in the Editor and select **Show Arguments**. You can then perform the edit operation in the **Text View** tab of the dialog box that opens. You cannot perform any edit operations in the Binary View tab.

Note that you perform edit operations on buffer data only, not on Snapshot data - which is read-only.



When you copy data from a buffer, VuGen allows you to copy the data either in hexadecimal format or in decimal format. When you insert data into a buffer, VuGen allows you to specify the format of the data—single byte, 2-byte, or 4-byte.

How to Record a Windows Sockets Script

This task describes how to set up a Windows Sockets recording and how to record the session.

1. **Open the recording options - optional**

   After creating a WinSock script, select **Record > Recording Options** and click the **WinSock** node.

2. **Select a translation table - optional**

   In the **EBCDIC section**, select a translation table. If your data is in ASCII format, select the **None** option—otherwise VuGen will convert the ASCII data. For details, see "Translation Tables" on page 791.

3. **Exclude any non-relevant sockets - optional**

   In the **Exclude Settings** section, add any non-relevant sockets to the list. You should exclude hosts and ports that do not influence the server load under test, similar to the local host and the DNS port (53), which are excluded by default.

   To exclude the entries from the recording, but include them in the log, clear the **Do not include excluded sockets in log** option.

   For user interface details, see the "WinSock Node" on page 331.

4. **Set a think time threshold - optional**

   Indicate a think time threshold. If VuGen detects a pause in action less than the threshold time, it will not generate a **Think Time** step/ **lr_think_time** function. For details, see the "WinSock Node" on page 331.

5. **Record the session**

   Record the session and save the script.

6. **Parameterize the script - optional**

   Replace recorded values with parameters using the shortcut menu. For more information, see "Parameters" on page 227

7. **Regenerate the script - optional**

   If you need to regenerate the script, for example if you want to include an excluded host:port, or if the translation was not correct:

   - Select **Record > Regenerate Script**.

   - Click the **Options** link.

   - Under **General**, select **Protocols**, and then under **Active Protocols**, ensure that the **Windows Sockets** checkbox is selected.

   - Under **Sockets**, select **Winsock**, and then modify the settings.

Note: Options for script regeneration are available for multi-protocol scripts only.

How to View and Modify Windows Sockets Buffers

The following steps describe how to view, modify, and navigate through WinSock buffer data.

Modifying buffer data

You can modify buffer data in the Show Arguments dialog box for specific **lrs** steps in a Vuser script. You can use the Show Arguments dialog box to modify buffer data for the following steps:

- lrs_length_receive
- lrs_length_send
- lrs_receive
- lrs_receive_ex
- lrs_send

For further details on these steps, see the Function Reference (**Help > Function Reference**).

To display the Show Arguments dialog box for any of the above steps, right-click a step in the Editor and select **Show Arguments**. A dialog box opens and displays the buffer data in the **Buffer Content** section of the dialog box.

For further details about editing buffer data in Windows Sockets steps, see "Buffer Data Editing" on page 795.

Note that you cannot modify any data in the Snapshot pane

View and modify the data in the data.ws file

In the Solution Explorer, double-click the **data.ws** file. The contents of the data.ws file appear in the VuGen Editor. Modify the data directly in the Editor. For details, see "Windows Sockets Data" on page 792.

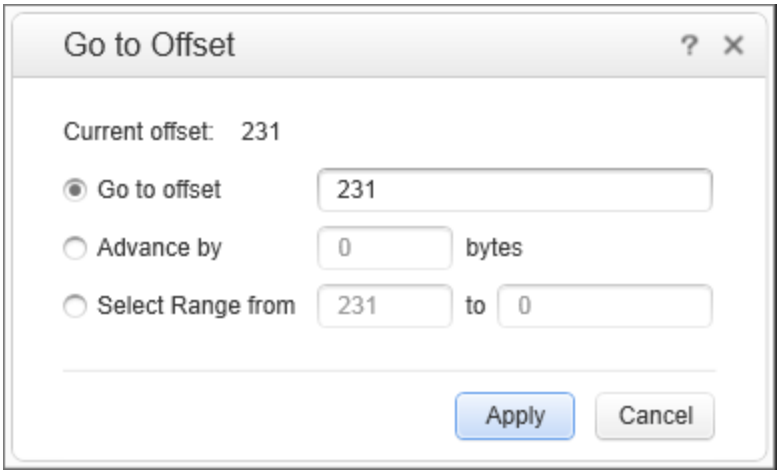
Note that it is possible to modify data.ws files, it is recommended that you do not modify these files.

View the data in the Snapshot pane

1. Ensure that the Snapshot pane is displayed.
2. Open the Vuser script in the Editor and select a step, or double-click an entry in the Step Navigator. The associated snapshot is displayed in the Snapshot pane. You cannot edit the snapshot data.

Navigate within the snapshot data

To navigate within the buffer data, display in the Snapshot pane, and then click **Go to**. The Go to Offset dialog box opens.



- To go to a specific offset within the buffer (absolute), select **Go to offset** and specify an offset value. Click **Apply**.
- To jump to a location relative to the cursor, click **Advance by** and specify the number of bytes to advance. To advance ahead, enter a positive value. To move backwards within the buffer, use a negative value. Click **Apply**.
- To select a range of data within the buffer, click **Select range from** and specify the beginning and end offsets. Click **Apply**.

Insert data into a buffer

You can insert a numerical value into a data buffer. You can insert the data as a single, double-byte, or 4-byte value. The following steps describe how to insert a number into a data buffer.

1. Copy the numerical data to be inserted to the clipboard.
2. Right-click a step in the Editor and select **Show Arguments**.
3. In the dialog box that opens, ensure that the **Text View** tab is displayed.
4. Under **Buffer Content**, right-click at the location in the buffer where you want to insert the data, and then select **Advanced > Paste Byte** or **Advanced > Paste Short (2 bytes)** or **Advanced > Paste Long (4 bytes)**.
5. Click **OK**. VuGen inserts the data into the buffer.

Copy and paste blocks of data

You can modify the buffer data as characters, decimal numbers, or hexadecimal numbers. For details, see "Buffer Data Editing" on page 795.

Note that you can edit buffer data only when you view the step arguments - you cannot edit buffer data in the Snapshot pane.

1. Right-click a step in the Editor and select **Show Arguments**. A dialog box opens and displays the buffer data in the **Buffer Content** section of the dialog box.
2. To copy buffer data:
   - As characters, select one or more bytes and press Ctrl+c.

- As a decimal number, select one or more bytes, right-click in the selection and select **Advanced > Copy As Decimal Number**.
- As a hexadecimal number, select one or more bytes, right-click in the selection and select **Advanced > Copy As Hexadecimal Number**.

3. To paste the data:
   - As characters, press Ctrl+V.
   - As a single byte (assuming the size of the data on the clipboard is a single byte), right-click at the desired location in the buffer and click **Advanced > Paste Byte**.
   - In short format (2-byte), right-click at the desired location in the buffer and click **Advanced > Paste Short (2 bytes)**.
   - In long format (4-byte), right-click at the desired location in the buffer and click **Advanced > Paste Long (4 bytes)**.
4. To delete data, select the data in the Text view, right-click inside the selection and select **Delete**.

Data Buffers

When you use VuGen to create a Windows Sockets Vuser script, VuGen creates the **data.ws** data file. This file contains the data that is transmitted, and is expected to be received, during the replay sessions. You can right-click any step in the Editor and then select **Show Arguments** to show the buffer content that is stored in **data.ws** for the selected step. Using the **Text View** tab of the dialog box that opens, you can edit the data that is stored for any data buffer.

The **data.ws** data file has the following format:

- File header
- A list of buffers and their contents

The file header includes an internal version number of the data file format. The current version is 2. If you try to access data from a data file with format version 1, VuGen issues an error.

```
;WSRData 2 1
```

An identifier precedes each record, indicating whether the data was received or sent, followed by the buffer descriptor, and the number of bytes received (for **lrs_receive** only). The buffer descriptor contains a number identifying the buffer.

For example,

```
recv buf5   25
```

indicates that the buffer contains data that was received. The record number is 5, indicating that this receive operation was the sixth data transfer (the index is zero based), and twenty-five bytes of data were received.

If your data is in ASCII format, the descriptor is followed by the actual ASCII data that was transferred by the sockets.

If your data is in EBCDIC format, it must be translated through a look-up table. For information on setting the translation table, see the "WinSock Node" on page 331. The EBCDIC whose ASCII

value (after translation) is printable, is displayed as an ASCII character. If the ASCII value corresponds to a non-printable character, then VuGen displays the original EBCDIC value.

```
recv buf6 39
    "\xff\xfb\x01\xff\xfb\x03\xff\xfd\x01"
    "\r\n"
    "SunOS UNIX (sunny)\r\n"
```

The following segment shows the header, descriptors, and data in a typical data file:

```
;WSRData 2 1
send buf0 48
    "\xff\xfd\x01\xff\xfd\x03\xff\xfb\x03\xff\xfb\x18"
recv buf1 15
    "\xff\xfd\x18\xff\xfd\x1f\xff\xfd"
    "#"
    "\xff\xfd"
    "'"
    "\xff\xfd"
    "$"
send buf2 24
    "\xff\xfb\x18"
```

Go To Offset Dialog Box

This dialog box allows you to go to a specific location within the recorded data.

To access	On the Snapshot pane toolbar, click **Go to**.
Relevant tasks	"How to View and Modify Windows Sockets Buffers" on page 797

User interface elements are described below:

UI Element	Description
Current offset	The current offset of the cursor (read only).
Go to offset	Goes to a specific, absolute offset within the data.
Advance by...bytes	Jumps to a location relative to the cursor, by a number of bytes. Positive values indicate a forward direction. Negative values indicate a reverse direction.
Select range from...to...	Selects a range of data within the buffer.
Apply	Moves the cursor to the specified offset.

Wireless Protocols

WAP Protocol Overview

The Wireless Application Protocol (WAP) is an open, global specification that enables mobile users with wireless devices to instantly access and interact with information and services.

The WAP protocol specifies a microbrowser thin-client using a new standard called WML that is optimized for wireless handheld mobile terminals. WML is a stripped-down version of XML.

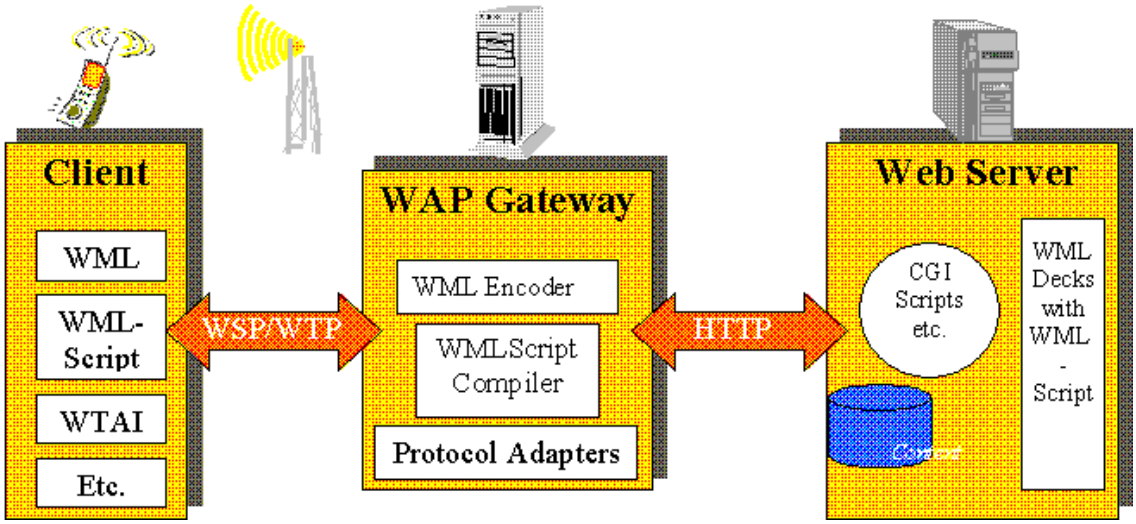
WAP also specifies a proxy server that:

- Acts as a gateway between the wireless network and the wire-line Internet
- Provides protocol translation
- Optimizes data transfer for the wireless handset

WAP architecture closely resembles the WWW model. All content is specified in formats that are similar to the standard Internet formats. Content is transported using standard protocols in the WWW domain and an optimized HTTP-like protocol in the wireless domain (Wireless Session Protocol). You locate all WAP content using WWW standard URLs.

WAP uses many WWW standards, including authoring and publishing methods. WAP enhances some of the WWW standards in ways that reflect the device and network characteristics. WAP extensions are added to support Mobile Network Services such as Call Control and Messaging. It accounts for the memory and CPU processing constraints that are found in mobile terminals. WAP also supports low bandwidth and high latency networks.

WAP assumes the existence of a gateway that is responsible for encoding and decoding data transferred to and from the mobile client. The purpose of encoding content delivered to the client is to minimize the size of data sent to the client over-the-air, as well as to minimize the computational energy required by the client to process that data. The gateway functionality can be added to origin servers, or placed in dedicated gateways as illustrated below.



WAP Toolkits

To assist developers in producing WAP applications and services, the leading companies such as Nokia, Ericsson, and Phone.com, have developed toolkits. The WAP Toolkit provides an environment for developers who want to provide Internet services and content for mobile terminals. It allows developers to write, test, debug, and run applications on a PC-based simulator phone. The toolkit allows users to browse WAP sites through an HTTP connection or a WAP gateway.

A mobile phone communicates with a gateway in WSP protocol; a toolkit can communicate with the gateway, or directly with the server. VuGen automatically detects the communication mode that is configured in the toolkit: WSP or HTTP. If you are interested in the traffic to the gateway, you record in WSP mode. If you want to check the server and the content providers, you can record your toolkit session in HTTP mode, and bypass the gateway.

VuGen uses custom API functions to emulate a user session. Most functions are the standard Web protocol functions utilizing the HTTP protocol. Several WAP functions emulate actions specific to WAP Vusers.

Push and Pull Technology

In the normal client/server model, a client requests information or a service from a server. The server responds by transmitting information or performing a service to the client. This is known as **pull** technology—the client pulls information from the server.

In contrast to this, there is also **push** technology. The WAP push framework transmits information to a device without a previous user action. This technology is also based on the client/server model, but there is no explicit request from the client before the server transmits its content.

To perform a push operation in WAP, a **Push Initiator** (PI) transmits content to a client. However, the Push Initiator protocol is not fully compatible with the WAP Client—the Push Initiator is on the Internet, and the WAP Client is in the WAP domain. Therefore, we need to insert a translating gateway to serve as an intermediary between the Push Initiator and the WAP Client. The translating gateway is known as the **Push Proxy Gateway** (PPG).

The access protocol on the Internet side is called the **Push Access Protocol** (PAP).

The protocol on the WAP end is called the Push **Over-The-Air** (OTA) protocol.

The Push Initiator contacts the Push Proxy Gateway (PPG) over the Internet using the PAP Internet protocol. PAP uses XML messages that may be tunneled through various well-known Internet protocols such as HTTP. The PPG forwards the pushed content to the WAP domain. The content is then transmitted using the OTA protocol over the mobile network to the destination client. The OTA protocol is based on WSP services.

In addition to providing basic proxy gateway services, the PPG is capable of notifying the Push Initiator about the final status of the push operation. In two-way mobile networks, it can also wait for the client to accept or reject the content.

Push Services Types

Push services can be of the SL or SI type:

- **SL.** The Service Loading (SL) content type provides the ability to cause a user agent on a mobile

client to load and execute a service—for example, a WML deck. The SL contains a URI indicating the service to be loaded by the user agent without user intervention when appropriate.

- **SI.** The Service Indication (SI) content type provides the ability to send notifications to end-users in an asynchronous manner. For example, the notifications may be about new emails, changes in stock price, news headlines, and advertising.

In its most basic form, an SI contains a short message and a URI indicating a service. The message is presented to the end-user upon reception, and the user is given the choice to either start the service indicated by the URI immediately, or postpone the SI for later handling. If the SI is postponed, the client stores it and the end-user is given the ability to act upon it at a later point of time.

VuGen Push Support

Push support for VuGen is divided into three parts:

- Push support at the client end—the ability to accept push messages.
- Push support to WAP HTTP Vusers—emulating Push Initiators.
- Push messages (SI =; SL) format services—formatting push messages.

Client Push Support

At the client end, VuGen supports both push services (SL and SI) for all replay modes (CO and CL). The **wap_wait_for_push** function instructs the Vuser to wait for a push message to arrive. You set the timeout for this function in the run-time settings.

When a push message arrives, the Vuser parses it to determine its type and to retrieve its attributes. If parsing was successful, it generates and executes a pull transaction to retrieve the relevant data. You can disable the pull event, indicating to the Vuser not to retrieve the message data by configuring the run-time settings.

Emulating Push Initiators

Push support for WAP HTTP Vusers enables you to perform load testing of the PPG. Push support allows Vusers to function as Push Initiators supporting the **Push Access Protocol** (PAP). The PAP defines the following sets of operations between the PI and the PPG:

- Submit a Push request.
- Cancel a Push request.
- Submit a query for the status of a push request.
- Submit a query for the status of a wireless device's capabilities.
- Initiate a result notification message from the PPG to the PI.

All operations are request/response—for every initiated message, a response is issued back to the PI. PI operations are based on the regular HTTP POST method supported by VuGen. Currently, only the first two operations are supported through **wap_push_submit** and **wap_push_cancel.**

Formatting Push Messages

You can submit data to a Web server using the **web_submit_data** function. It is difficult, however, to send long and complex data structures using this function. To overcome this difficulty and

provide a more intuitive API function, several new API functions were added to properly format the XML message data: **wap_format_si_msg** and **wap_format_sl_msg**. For more information about these functions, see the Function Reference (**Help > Function Reference**).

MMS (Multimedia Messaging Service) Protocol Overview

MMS (Multimedia Messaging Service) is an extension of the SMS protocol. Whereas SMS messages can contain only text, MMS allows you to send and receive messages with a wide range of content to MMS capable handsets. This content can be in the form of text, sound, email messages, images, video clips, and even streaming data. It is also possible to send multimedia messages from a mobile phone to an email address.

An MMS message typically includes a collection of attachments. While SMS messages are limited to 160 bytes, an MMS message could be several MBs in size. MMS usually requires a third generation (3G) network to enable such large messages to be delivered.

To receive an MMS message, a mobile phone receives an MMS notification over SMS. The SMS message can be received over various SMS protocols such as SMPP, UCP, and CIMD2. The SMS message contains a unique path to the MMS message stored in the MMSC server's database and the mobile phone uses this path to download the message from the SMSC. The current version of VuGen supports the receiving of MMS notifications over the SMPP interface.

Multimedia Messaging Service Vuser scripts support the 1.0 and 1.1 versions of the MMS protocol, as defined by OMA (Open Mobile Alliance organization). Using MMS Vusers, you can send MMS messages to the MMSC server directly over the HTTP protocol, or over the WAP protocol through a WAP gateway.

Multimedia Messaging Service functions emulate the sending and receiving of MMS messages. Each function begins with an **mm** prefix. For detailed syntax information for these functions, see the Function Reference (**Help > Function Reference**).

How to Run an MMS Scenario in the Controller

An MMS (Multimedia Messaging Service) scenario requires a command line setting.

To set the MMS command line setting:

1. From the Scenario Schedule screen, click **Details**. The Group Information dialog is displayed.
2. If the Command line box is not visible, click the **More** button.
3. Add the following to the end of the Command line text: `-usingwininet yes`
4. Click **OK** to accept the Command line switch.

Advanced Topics

Manually Programming a Script using the VuGen Editor

Manually Programming Scripts - Overview

VuGen allows you to program your own functions into the script, instead of recording an actual session. You can use the Vuser API or standard programming functions. Vuser API functions allow you to gather information about Vusers. For example, you can use Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test or monitoring.

This chapter describes how to program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes.

You can also develop a Vuser script through programming within the Visual C and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API function libraries. For more information, see "Creating Scripts with Visual Studio" on page 809.

To create a customized script, you first create a skeleton script. The skeleton script contains the three primary sections of a script: **init**, **actions**, and **end**. These sections are empty and you manually insert functions into them.

You can create empty scripts for the following programming languages:

- C
- Java
- Visual Basic
- VBScript
- JavaScript

Note: When working with JavaScript and VBScript Vusers, the COM objects that you use within your script must be fully automation compliant. This makes it possible for one application to manipulate objects in another application, or to expose objects so that they may be manipulated.

JavaScript Vusers

You can create an empty JavaScript Vuser script, in which to place JavaScript code. This script type lets you incorporate your existing JavaScript application into VuGen. To create an empty JavaScript Vuser script, select **JavaScript Vuser** from the list of protocols in the Create a New Script dialog box.

```
function Actions()
{
      //"TO DO: Place your business process/action code here
      return(lr.PASS);
}
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a JavaScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the `TO DO` comment.

An additional section that is viewable from VuGen, is the **global.js** file, which creates the objects for the Vuser API functions and the Javascript. For example, the following code creates the standard object **lr**:

```
var lr = new ActiveXObject("LoadRunner.LrApi")
```

C Vuser Scripts

In a C Vuser script, you can use any C code that conforms to the standard ANSI conventions. To create an empty C Vuser script, select **C Vuser** in the Create a New Script dialog box. VuGen creates an empty C Vuser script:

```
Action1()
{
    return 0;
}
```

You can use C Vuser functions in all of Vuser script types that use C functions.

See the Function Reference (**Help > Function Reference**) for a C reference that includes syntax and examples of commonly used C functions.

Guidelines for Using C Functions

All standard ANSI-C conventions apply to C Vuser scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other C programs. You declare and define variables using ANSI C conventions.

The C interpreter that is used to run Vuser scripts accepts the standard ANSI C language. It does not support any Microsoft extensions to ANSI C.

Before you add any C functions to a Vuser script, note the following limitations:

- A Vuser script cannot pass the address of one of its functions as a callback to a library function.
- The **stdargs**, **longjmp**, and **alloca** functions are not supported in Vuser scripts.
- Vuser scripts do not support structure arguments or return types. Pointers to structures are supported.
- In Vuser scripts, string literals are read-only. Any attempt to write to a string literal generates an access violation.
- C Functions that do not return int, must be casted. For example,
  `extern char * strtok();`

Calling libc Functions

In a Vuser script, you can call **libc** functions. However, since the interpreter that is used to run Vuser scripts does not support any Microsoft extensions to ANSI C, you cannot use Microsoft's include files. You can either write your own prototypes, or ask HP Customer Support to send you ANSI-compatible include files containing prototypes for **libc** functions.

Linking Mode

The C interpreter that is used to run Vuser scripts uses a "lazy" linking mode in the sense that a function need not be defined at the start of a run, as long as the function is defined before it is used. For example:

```
lr_load_dll("mydll.dll");
    myfun();  /* defined in mydll.dll -- can be called directly,
                 immediately after myfun.dll is loaded. */
```

VBScript Vusers

You can create an empty VBScript Vuser script, in which you can place VBScript code. This script type lets you incorporate your VBScript application into VuGen. To create an empty VBScript Vuser script, select VB Script Vuser from the list of protocols in the Create a New Script dialog box. VuGen creates an empty VBScript Vuser script:

```
Public Function Actions()
      `"TO DO: Place your action code here
      Actions = lr.PASS
End Function
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a VBScript function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the `TO DO` comment.

An additional section that is viewable from VuGen, is the **global.vbs** file, which creates the objects for the Vuser API functions and VB Script. For example, the following code creates the standard object **lr**:

```
Set lr = CreateObject("LoadRunner.LrApi")
```

Java Vusers

In Java Vuser scripts, you can place any standard Java code. To create an empty Java Vuser script, select Java Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty Java script:

```
import lrapi.lr;
public class Actions
{
    public int init() {
        return 0;
    }
    public int action() {
        return 0;
    }
    public int end() {
        return 0;
    }
}
```

Note that for Java type Vusers, you can only edit the **Actions** class. Within the Actions class, there are three methods: **init**, **action**, and **end**. Place initialization code in the **init** method, business processes in the **action** method, and cleanup code in the **end** method.

VB Vusers

You can create an empty Visual Basic Vuser script, in which you can place Visual Basic code. This script type lets you incorporate your Visual Basic application into VuGen. To create an empty VB Vuser script, select VB Vuser from the Custom category, in the New Virtual User dialog box. VuGen creates an empty VB script:

```
Public Function Actions() As Long
      `"TO DO: Place your action code here
      Actions = lr.PASS
End Function
```

VuGen creates three sections, **vuser_init**, **action**, and **vuser_end**. Each of these sections contain a VB function—**Init**, **Actions**, and **Terminate** respectively. You place your code within these functions, as indicated by the `TO DO` comment.

An additional section that is viewable from VuGen, is the **global.vba** file, which contains the object and variable global declarations for Vusers and the VB application.

Replay Error with VB Vuser Scripts

If you are getting error number -25210 when trying to replay a VB Vuser script, you may have a problem with some of your DLL files.

Solution

1. Open the **c:\Program Files\Common Files\Microsoft Shared\vba\vba6** folder.
2. Locate the **VBE6.dll** and **VBE6EXT.OLB** files.
3. Right click the files and click properties to see the version of each file.
4. If either the **VBE6.dll** or the **VBE6EXT.OLB** file versions are between 6.04.9972 and 6.05.1024, they both must be replaced. If neither file version is in this range, contact HP software support.
5. Replace the **VBE6.dll** file with version 6.04.9972 or 6.05.1024.
6. Replace the **VBE6EXT.OLB** file with version 6.04.9969 or 6.05.1024.

.NET Vusers

You can create an empty .NET Vuser script, in which to place .NETcode. This script type lets you incorporate your existing .NET application into VuGen. To create an empty .NET Vuser script, select .NET Vuser from the Custom category, in the Create New Script dialog box.

In a .NET vuser script the default language is C#. If your script is generated from a recorded session, VuGen enables you to change the script language from C# to VB.NET by selecting Visual Basic .NET Language from **Record > Recording Options > General > Script** and regenerating the script.

> **Tip:** You can edit the script in Visual Studio by clicking the  button.

The following example is in C#:

```
namespace Script
                        {
                        public class VuserClass
                        {
                        public int vuser_init()
                        {
                        return 0;
                        }
                        public int Action()
                        {
                        return 0;
                        }
                        public int vuser_end()
                        {
                        return 0;
                        }
                        }
                }
```

Note: Place initialization code in the **init** method, business processes in the **action** method, and cleanup code in the **end** method.

Creating Scripts with Visual Studio

Creating Vuser Scripts in Visual Studio - Overview

There are several ways to create Vuser scripts: through VuGen or a development environment such as Visual Studio.

VuGen	You can use VuGen to create Vuser script that run on Windows or Linux platforms by recording or by manually programming within the VuGen editor. You create the script in a Windows environment and run it in either Windows or Linux—recording is not supported on Linux.
Visual Studio	For users working with Visual Studio, you can program in Visual Basic, C, C# or C++. The programs must be compiled into a dynamic link library (dll).

This chapter describes how to develop a Vuser script through programming within the Visual C, Visual C# and Visual Basic environments. In these environments, you develop your Vuser script within your development application, while importing the Vuser API libraries.

You can also program a Vuser script from within the VuGen editor, incorporating your application's libraries or classes. Programming within VuGen is available for C, Java, Visual Basic, VBScript, and JavaScript. For more information, see "Manually Programming a Script using the VuGen Editor" on page 804.

To create a Vuser script through programming, you can use a VuGen template as a basis for a larger Vuser script. The template provides:

- correct program structure
- Vuser API calls
- source code and makefiles for creating a dynamic library

An online C reference of the common functions used in Vuser scripts, are included in the Function Reference (**Help > Function Reference**).

How to Create a Vuser Script with Visual C

You can create Vuser scripts using Visual C version 6.0 or higher.

1. In Visual C, create a new project - dynamic link library (dll). Select **File > New** and click the Projects tab.
2. In the Wizard, select **empty dll**.
3. Add the following files to the project:
   - A new *cpp* file with 3 exported function: *init*, *run*, *end* (the names may be customized).
   - The library file lrun50.lib (located in the <lr installation dir>/lib).
4. In the project settings change the following:
   - Select the C/C++ tab and select **Code generation** (Category) **> Use Run Time library** (List). Change it to: **Multithreaded dll.**
   - Select the C/C++ tab and select **Preprocessor** (Category) > **Preprocessor definitions** (edit field) Remove _DEBUG.
5. Add code from your client application, or program as you normally would.
6. Enhance your script with Vuser API functions. For example, **lr_output_message** to issue messages, **lr_start_transaction** to mark transactions, and so forth. For more information, see the General functions in the Function Reference (**Help > Function Reference**).
7. Build the project. The output will be a DLL.
8. Create a folder with the same name as the DLL and copy the DLL to this folder.
9. In the **lrvuser.usr** file in the *Template* folder, Update the USR file key *BinVuser* with the DLL name: `BinVuser=<DLL_name>`.

   Example:

   In the following example, the **lr_output_messsage** function issues messages indicating which section is being executed. The **lr_eval_string** function retrieves the name of the user. To use the following sample, verify that the path to the Vuser API include file, **lrun.h** is correct.

```
#include "c:\lrun_5\include\lrun.h"
extern "C" {
int __declspec(dllexport) Init (void *p)
{
    lr_output_message("in init");
```

```
return 0;
}
int __declspec(dllexport) Run (void *p)
{
    const char *str = lr_eval_string("<name>");
    lr_output_message("in run and parameter is %s", str);
return 0;
}
int __declspec(dllexport) End (void *p)
{
    lr_output_message("in end");
return 0;
}
} //extern C end
```

How to Create a Vuser Script with Visual Basic or Visual C#

Create a Vuser in Visual Basic or Visual C#

1. In Microsoft Visual Basic or in Microsoft Visual C#, create a new project by selecting **File > New Project** > **LoadRunner Virtual User**. A new project is created with one class and a template for a Vuser.
2. Save the project by selecting **File > Save Project**.
3. Open the Object Browser (View menu). Select the LoadRunner Vuser library and double-click on the Vuser Class module to open the template. The template contains three sections, **Initialize**, **Actions**, and **Terminate**.
   - Visual Basic template

```
Option Explicit
Implements Vuser
Private Sub Initialize()
'Implement the Vuser initialization code here
End Sub
Private Sub Actions()
'Implement the Vuser main Action code here
End Sub
Private Sub Terminate()
'Implement the Vuser termination code here
End Sub
```

   - Visual C# template

```
public int Initialize()
        {
            // TO DO: Add virtual user's initialization routines
            return lr.PASS;
        }
        public int Actions()
```

```
{
    // TO DO: Add virtual user's business process actions
    return lr.PASS;
}
public int Terminate()
{
    // TO DO: Add virtual user's termination routines
    return lr.PASS;
}
```

4. Add code from your client application, or program as you normally would.
5. Use the Object Browser to add the desired VuGen elements to your code such as transactions, think time, rendezvous points, and messages.
6. Enhance your program with run-time settings and parameters. For more information, see "How to Configure Run-Time Settings and Parameters" below.
7. Build the Vuser script by selecting **Build > Build Solution** *project_name*.dll.

The project is saved in the form of a Vuser script (.usr) in the same folder as the project.

How to Configure Run-Time Settings and Parameters

After you create the DLL for your script, you create a script (*.usr*) and configure its settings. The *lrbin.bat* utility provided with VuGen lets you define parameters and configure runtime settings for scripts created with Visual C and Basic. This utility is located in the *bin* folder of the product installation.

Configure Runtime Settings and Parameterize Scripts

1. In the product's *bin* folder, double-click on *lrbin.bat*. The Standalone Vuser Configuration dialog box opens.
2. Select **File > New**. Specify a script name for the *usr* file. The script name must be identical to the name of the folder to which you saved the DLL.
3. Select **Vuser > Advanced** and enter the DLL name in the Advanced dialog box.
4. Select **Replay > Run-time Settings** to define run-time settings. The Run-time Settings dialog box is identical to that displayed in the VuGen interface. For more information, see "Run-Time Settings" on page 332.
5. Select **Vuser > ParameterList** to define parameters for your script. The Parameter dialog boxes are identical to those in VuGen. For more information, see "Parameters" on page 227.

Test the script by running it in standalone mode. Select **Replay > Run Vuser**. The Vuser execution window appears while the script runs.

Language Support

Language Support - Overview

VuGen supports multilingual environments, allowing you to use languages other than English on native language machines when creating and running scripts.

When working with languages other than English, the primary issue is ensuring that VuGen recognizes the encoding of the text during record and replay. The encoding applies to all texts used by the script. This includes texts in HTTP headers and HTML pages for Web Vusers, data in parameter files, and others.

Windows 2000 and higher lets you save text files with a specific encoding directly from Notepad: ANSI, Unicode, Unicode big endian, or UTF-8.

By default, VuGen works with the local machine encoding (ANSI). Some servers working with foreign languages, require you to work with UTF-8 encoding. To work against this server, you must indicate in the Advanced recording options, that your script requires UTF-8 encoding.

Page Request Header Language

Before running a Web script, you can set the page's request header to match your current language. In the Internet Protocol run-time settings, you set the language of the *Accept-Language* request header. This header provides the server with a list of all of the accepted languages.

To set this value, select **Replay > Run-Time Settings > Internet Protocol > Preferences > Advanced > Options > Accept-Language request header** and select the desired language.

For user interface details, see "Internet Protocol > Preferences Node" on page 364.

How to Convert Encoding Format of a String

You can manually convert a string from one encoding to another (UTF-8, Unicode, or locale machine encoding) using the **lr_convert_string_encoding** function with the following syntax:

```
lr_convert_string_encoding(char * sourceString, char * fromEncoding,
char * toEncoding, char * paramName)
```

The function saves the result string (including its terminating NULL) in the third argument, *paramName*. It returns a 0 on success and -1 on failure.

The format for the **fromEncoding** and **toEncoding** arguments are:

LR_ENC_SYSTEM_LOCALE	NULL
LR_ENC_UTF8	"utf-8"
LR_ENC_UNICODE	"ucs-2"

In the following example, **lr_convert_string_encoding** converts "Hello world" from the system locale to Unicode.

```
Action()
{
    int rc = 0;
    unsigned long converted_buffer_size_unicode = 0;
    char               *converted_buffer_unicode = NULL;
    rc = lr_convert_string_encoding("Hello world", NULL, LR_ENC_
UNICODE, "stringInUnicode");
    if(rc < 0)
    {
        // error
    }
return 0;
}
```

In the replay log, the output window shows the following information:

```
Output:
Starting action Action.
Action.c(7): Notify: Saving Parameter "stringInUnicode =
H\x00e\x00l\x00l\x00o\x00 \x00w\x00o\x00r\x00l\x00d\x00\\x00"
Ending action Action.
```

The result of the conversion is saved to the *paramName* argument.

How to Convert Encoding Format of Parameter Files

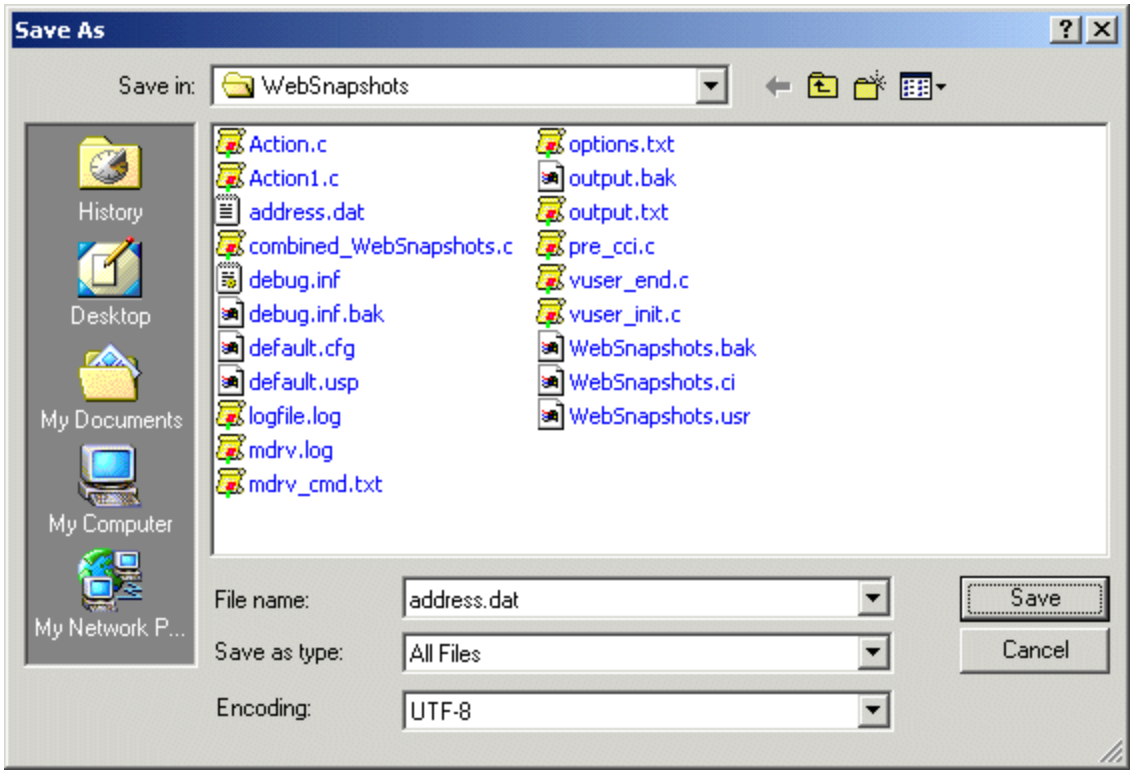
The parameter file contains the data for parameters that were defined in the script. This file, stored in the script's folder, has a **.dat* extension. When running a script, Vusers use the data to execute actions with varying values.

By default, VuGen saves the parameter file with your machine's encoding. When working with languages other than English, however, in cases where the server expects to receive the string in UTF-8, you may need to convert the parameter file to UTF-8. You can do this directly from Notepad, provided that you are working with Windows 2000 or higher.

Apply UTF-8 Encoding to a Parameter File

1. Select **Vuser > Parameter List** and view the parameter properties.
2. In the right pane, locate the parameter file in the **File path** box.
3. With the parameter table in view, click **Edit in Notepad**. Notepad opens with the parameter file in csv format.
4. In the **Save as type** box, select *All Files*.

   In the **Encoding** box, select *UTF-8* type encoding.



5. Click **Save**. Notepad asks you to confirm the overwriting of the existing parameter file. Click **Yes**.

   VuGen now recognizes the parameter file as UTF-8 text, although it still displays it in regular characters.

How to Record Web Pages with Foreign Languages

When working with Web or other Internet protocols, you can indicate the encoding of the Web page text for recording. The recorded site's language must match the operating system language. You cannot mix encodings in a single recording—for example, UTF-8 together with ISO-8859-1 or shift_jis.

This task describes how to record web pages with foreign languages using VuGen.

Automatically Record Foreign Language Web Pages.

In order to be recognized as a non-English Web page, the page must indicate the charset in the HTTP header or in the HTML meta tag. Otherwise, VuGen will not detect the EUC-JP encoding and the Web site will not be recorded properly. To instruct VuGen to record non-English requests as **EUC-JP** or **UT-8**, select **Record > Recording Options > HTTP Properties > Advanced > support charset** and select the appropriate option in the Recording Options dialog box, **HTPP Properties: Advanced** node. For user interface details, see "HTTP > Advanced Node" on page 297.

Note that by selecting the **EUC-JP** or **UTF-8** option in the Recording Options, you are forcing VuGen to record a Web page with the selected encoding, even when it uses different encoding. If, for example, a non-EUC encoded Web page is recorded as EUC-JP, the script will not replay properly.

Manually Record Foreign Language Web Pages

You can manually add full support for recording and replaying of HTML pages encoded in EUC-JP using the **web_sjis_to_euc_param** function. This also allows VuGen to display Japanese EUC-encoded characters correctly in Vuser scripts.

When you use **web_sjis_to_euc_param**, VuGen shows the value of the parameter in the Execution Log using EUC-JP encoding. For example, when you replay the **web_find** function, VuGen displays the encoded values. These include string values that were converted into EUC by the **web_sjis_to_euc_param** function, or parameter substitution when enabled in the **Run-Time Setting > Log > Extended Log**.

Foreign Languages - Troubleshooting and Limitations

This section describes troubleshooting and limitations when working with foreign languages.

Browser Configuration

If, during recording, non-English characters in the script are displayed as escaped hexadecimal numbers (For example, the string "　=;" becomes "%DC%26"), you can correct this by configuring your browser not to send URLs in UTF-8 encoding. In Internet Explorer, select **Tools > Internet Options and** click the **Advanced** tab. Clear the **Always Send URLs as a UTF-8** option in the Browsing section.

For more information about **web_sjis_to_euc_param**, see the LoadRunner Function Reference.

Protocol Limitations

SMTP Protocol

If you work with SMTP protocol through MS Outlook or MS Outlook Express, the Japanese text recorded in a Vuser script is not displayed correctly. However, the script records and replays correctly.

Script Name Length

When recording in COM, FTP, IMAP, SMTP, POP3, or REAL mode, the length of the script name is limited to 10 multi-byte characters (21 bytes).

Application Lifecycle Management Integration

To open a script saved in an Application Lifecycle Management project from VuGen, or a scenario saved in an ALM project from Controller, add a new Test Set named "Default" (in English) to the ALM project.

Advanced Topics

Calling External Functions in DLLs

You can call functions that are defined in external DLLs. By calling external functions from your script, you can reduce the memory footprint of your script and the overall run-time.

To call the external function, you load the DLL in which the function is defined.

You can load a DLL in one of the following ways:

- Locally (for one script) by using the **lr_load_dll** function. For task details, see "How to Load a DLL Locally" on page 823.
- Globally (for all scripts) by adding statements to the **vugen.dat** file. For task details, see "How to Load a DLL Globally" on page 824.

Logging CtLib Server Messages

When you run a CtLib Vuser script, (Sybase CtLib, under the Client Server type protocols), all messages generated by the CtLib client are logged in the standard log and in the output file. By default, server messages are not logged. To enable logging of server messages (for debugging purposes), insert the following line into your Vuser script:

```
LRD_CTLIB_DB_SERVER_MSG_LOG;
```

VuGen logs all server messages in the Standard log.

To send the server messages to the output (in addition to the Standard log), type:

```
LRD_CTLIB_DB_SERVER_MSG_ERR;
```

To return to the default mode of not logging server errors, type the following line into your script:

```
LRD_CTLIB_DB_SERVER_MSG_NONE;
```

Note: Activate server message logging for only a specific block of code within your script, since the generated server messages are long and the logging can slow down your system.

Recording OLE Servers

VuGen currently does not support recording for OLE applications. These are applications where the actual process is not launched by the standard process creation routines, but by the OLE Automation system. However, you can create a Vuser script for OLE applications based on the following guidelines.

There are two types of OLE servers: executables, and DLLs.

DLL Servers

If the server is the DLL, it will eventually be loaded into the application process space, and VuGen will record the call to LoadLibrary. In this case, you may not even realize that it was an OLE application.

Executable Servers

If the server is the executable, you must invoke the executable in the VuGen in a special way:

- First, determine which process actually needs to be recorded. In most cases, the customer knows the name of the application's executable. If the customer doesn't know the name of the application, invoke it and determine its name from the NT Task Manager.

- After you identify the required process, click **Start Recording** in VuGen. When prompted for the Application name, enter the OLE application followed by the flag "/Automation". Next, launch the user process in the usual way (not via VuGen). VuGen records the running OLE server and does not invoke another copy of it. In most cases, these steps are sufficient to enable VuGen to record the actions of an OLE server.
- If you still are experiencing difficulties with recording, you can use the CmdLine program to determine the full command line of a process which is not directly launched. (The program is available in a knowledge base article on the Customer Support Web site, http://support.hp.com)

Using CmdLine

In the following example, CmdLine.exe is used to determine the full command line for the process MyOleSrv.exe, which is launched by some other process.

Determine its full command line

1. Rename *MyOleSrv.exe* to *MyOleSrv.orig.exe.*
2. Place *CmdLine.exe* in the same folder as the application, and rename it to *MyOleSrv.exe*.
3. Launch *MyOleSrv.exe*. It issues a popup with a message containing the complete command line of the original application, (including additional information), and writes the information into *c:\temp\CmdLine.txt*.
4. Restore the old names, and launch the OLE server, *MyOleSrv.exe*, from VuGen with the correct command line parameters. Launch the user application in a regular way - not through VuGen. In most cases, VuGen will record properly.

If you still are experiencing difficulties with recording, proceed with the following steps:

1. Rename the OLE server to MyOleSrv.1.exe, and CmdLine to MyOleSrv.exe.
2. Set the environment variables "CmdStartNotepad" and "CmdNoPopup" to 1. See "CmdLine Environment Variables" below for a list of the CmdLine environment variables.
3. Start the application (not from VuGen). Notepad opens with the full command line. Check the command line arguments. Start the application several times and compare the command line arguments. If the arguments are the same each time you invoke the application, then you can reset the CmdStartNotepad environment variable. Otherwise, leave it set to "1".
4. In VuGen, invoke the program, MyOleSrv.1.exe with the command line parameters (use Copy/Paste from the Notepad window).
5. Start the application (not from within VuGen).

CmdLine Environment Variables

You can control the execution of CmdLine through the following environment variables:

CmdNoPopup	If set, the popup window will not appear.
CmdOutFileName	If set, and non-empty, CmdLine will attempt to create this file instead of c:\temp\CmdLine.txt.
CmdStartNotepad	If set, the output file will be displayed in the notepad (Best used with CmdNoPopup).

Running a Vuser from the Linux Command Line

VuGen includes a Linux shell script utility, *run_db_vuser.sh*, that automatically performs the same operations as the virtual user but from the command line. It can perform each of the replay steps optionally and independently. This is a useful tool for debugging tests to be replayed on Linux.

Place the file run_db_vuser.sh in the $M_LROOT/bin folder. To replay a Vuser type:

```
run_db_vuser.sh vuser.usr
```

You can also use the following command line options:

-cpp_ only	This option will start the preprocessing phase. The output of this process is the file ` Vuser.c '.
-cci_ only	This option runs the compilation phase. The ` Vuser.c ' file is used as input, and the output produced is the ` Vuser.ci ' file.
-exec_ only	This option runs the Vuser, by taking as input the ` Vuser.ci ' file and running it via the replay driver.
-ci ci_ file	This option allows you to specify the name and location of a .ci file to be run. The second parameter contains the location of the .ci file.
-out output_ directory	This option allows you to determine the location of any output files created throughout the various processes. The second parameter is the directory name and location.
-driver driver_ path	This option allows you to specify the actual driver executable to be used for running the Vuser. By default the driver executable is taken from the settings in the VuGen.dat file.

Note that only one of the first three options can be used at a time for running the run_db_vuser.

Specifying the Vuser Behavior

Since VuGen creates the Vuser script and the Vuser behavior as two independent sources, you can configure user behavior without directly referencing the Vuser script, for example, wait times,

pacing times, looping iterations, logging, and so forth. This feature lets you make configuration changes to a Vuser, as well as store several `profiles' for the same Vuser script.

The ` Vuser.cfg ' file, by default, is responsible for defining this behavior - as specified in VuGen's Runtime settings dialog box. You can save several versions of this file for different user behavior and then run the Vuser script referencing the relevant *.cfg* file.

You can run the Vuser script with the relevant configuration file from a server machine. To do this, add the following to the Vuser command line:

```
-cfg c:\tmp\profile2.cfg
```

For information on command line parameters, see "Command Line Parameters" below.

Note that you cannot control the behavior file from VuGen. VuGen automatically uses the *.cfg* file with the same name as the Vuser. (You can, of course, rename the file to be ` Vuser.cfg '). However, you can do this manually from the command line by adding the -cfg parameter mentioned above to the end of the driver command line.

Note: The Linux utility, *run_db_vuser*, does not yet support this option.

Command Line Parameters

The Vusers can accept command line parameters when invoked. There are several Vuser API functions available to reference them (**lr_get_attrib_double**, and so on). In your environment, you can send command line parameters to the Vuser by adding them to the command line entry of the script window.

When running the Vuser from VuGen, you cannot control the command line parameters. You can do this manually, however, from the Windows command line by adding the parameters at the end of the line, after all the other driver parameters, for example:

```
mdrv.exe -usr c:\tmp\Vuser\Vuser.usr    -out c:\tmp\vuser     vuser_
command_line_params
```

Note: The Linux utility, *run_db_vuser*, does not yet support this option.

How to Create a New Vuser Type

The following steps describe how to create a new vuser type.

1. **Edit the mdrv.dat file**

   Edit the mdrv.dat file which resides in the M_LROOT\dat folder. Add a section for the new Vuser type with all of the applicable parameters from the following list.

   ```
   [<extension_name>]
   ExtPriorityType=< {internal, protocol}>
   WINNT_EXT_LIBS=<dll name for NT>
   WIN95_EXT_LIBS=<dll name for 95>
   SOLARIS_EXT_LIBS=<dll name for Solaris>
   ```

```
LINUX_EXT_LIBS=<dll name for Linux>
HPUX_EXT_LIBS=<dll name for HP>
AIX_EXT_LIBS=<dll name for IBM>
LibCfgFunc=<configuration function name>
UtilityExt=<other extensions list>
WINNT_DLLS=<dlls to load to the interpreter context, for NT>
WIN95_DLLS=<dlls to load to the interpreter context, for 95>
SOLARIS_DLLS=<dlls to load to the interpreter context, for Solaris>
LINUX_DLLS=<dlls to load to the interpreter context, for Linux>
HPUX_DLLS=<dlls to load to the interpreter context, for HP>
AIX_DLLS=<dlls to load to the interpreter context, for IBM>
ExtIncludeFiles=<extra include files. several files can be
seperated by a comma>
ExtCmdLineConc=<extra command line (if the attr exists concatenate
value)>
ExtCmdLineOverwrite=<extra command line (if the attr exists
overwrite value)>
CallActionByNameFunc=<interpreter exec_action function>
GetFuncAddress=<interpreter get_location function>
RunLogicInitFunc=<action_logic init function>
RunLogicRunFunc=<action_logic run function>
RunLogicEndFunc=<action_logic end function>
```

For example, an Oracle NCA Vuser type is represented by:

```
[Oracle_NCA]
ExtPriorityType=protocol
WINNT_EXT_LIBS=ncarp11i.dll
WIN95_EXT_LIBS=ncarp11i.dll
LINUX_EXT_LIBS=liboranca11i.so
SOLARIS_EXT_LIBS=liboranca11i.so
HPUX_EXT_LIBS=liboranca11i.sl
AIX_EXT_LIBS=liboranca11i.so
LibCfgFunc=oracle_gui_configure
UtilityExt=lrun_api,HttpEngine
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
SecurityRequirementsFiles=oracle_nca.asl
SecurityMode=On
```

VuGen was designed to be able to handle a new Vuser type with no code modifications. You may, however, need to add a special View.

There is no generic driver supplied with VuGen, but you can customize one of the existing drivers. To use a customized driver, modify *mdrv.dat*. Add a line with the platform and existing driver, then add a new line with your customized driver name, in the format <platform>_ DLLS=<my_replay.dll name>. For example, if your SAP replay dll is called SAPPLAY32.DLL, add the following two lines to the [sap] section of *mdrv.dat*:

```
WINNT=sapdrv32.exe
WINNT_DLLS=sapplay32.dll
```

2. ## Add a CFG file (optional)

   You can specify a configuration file to set the default run-time settings for your protocol. You define it in the LibCfgFunc variable in the mdrv.dat file, or place one called default.cfg in the new protocols subfolder under templates. A sample default.cfg follows.

```
[ThinkTime]
Options=NOTHINK
Factor=1
LimitFlag=0
Limit=1
[Iterations]
NumOfIterations=1
IterationPace=IterationASAP
StartEvery=60
RandomMin=60
RandomMax=90
[Log]
LogOptions=LogExtended
MsgClassData=0
MsgClassParameters=0
MsgClassFull=1
```

3. ## Insert an LRP file

   In the dat/protocols folder, insert an *lrp* file which defines the protocol. This file contains the configuration information for the protocol in the Protocol, Template, VuGen, and API sections. Certain protocols may have additional sections, corresponding to the additional run-time setting options.

   The Protocol section contains the name, category, description, and bitmap location for the protocol.

```
 [Protocol]
Name=WAP
CommonName=WAP
Category=Wireless
Description=Wireless Application Protocol - used for Web-based,
wireless communication between mobile devices and content
providers.
Icon=bitmaps\wap.bmp
Hidden=0
Single=1
Multi=0
```

   The Template section indicates the name of the various sections of the script and the default test name.

```
[Template]
vuser_init.c=init.c
vuser_end.c=end.c
Action1.c=action.c
Default.usp=test.usp
```

```
@@TestName@@.usr=wap.usr
default.cfg=default.cfg
```

The **VuGen** section has information about the record and replay engines, along with the necessary DLLs and run-time files.

The **API** section contains information about the protocol's script API functions.

You can use one of the existing lrp files in the protocols folder as a base for your new protocol.

4. ### Specify a Template

   After adding an *lrp* file, insert a subfolder to *M_LROOT/*template with a name corresponding to the protocol name defined in the *lrp* file. In this subfolder, insert a *default.cfg* file which defines the default settings for the general and run-time settings.

   If you want to use a global header file for all of your protocol's scripts, add a file named *globals.h*. This file should contain an include statement which points to a header file for the new protocol. For example, the template/http subfolder contains a file called *globals.h* which directs VuGen to the *as_web.h* file in the include folder:

   ```
   #include #as_web.h"
   ```

How to Load a DLL Locally

This task describes how to use the **lr_load_dll** function to load a DLL into your Vuser script. Once the DLL is loaded, you can call any function defined within the DLL without having to declare it in your script.

Call a function defined in a DLL

1. Use the **lr_load_dll** function to load the DLL at the beginning of your script. Place the statement at the beginning of the *vuser_init* section. **lr_load_dll** replaces the **ci_load_dll** function.

   Use the following syntax:

   ```
   lr_load_dll( library_name);
   ```

   Note that for Linux platforms, DLLs are known as shared libraries. The extension of the libraries is platform dependent.

2. Call the function defined in the DLL in the appropriate place within your script.

   In the following example, the **insert_vals** function, defined in **orac1.dll**, is called, after the creation of the **Test_1** table.

   ```
   int LR_FUNC Actions(LR_PARAM p)
   {
   lr_load_dll("orac1.dll");

   lrd_stmt(Csr1, "create table Test_1 (name char(15), id integer)\n",
   -1,
              1 /*Deferred*/, 1 /*Dflt Ora Ver*/, 0);
   lrd_exec(Csr1, 0, 0, 0, 0, 0);
   /* Call the insert_vals function to insert values into the table. */
   insert_vals();
   ```

```
lrd_stmt(Csr1, "select * from Test_1\n", -1, 1 /*Deferred*/, 1
/*Dflt Ora Ver*/, 0);
lrd_bind_col(Csr1, 1, =;NAME_D11, 0, 0);
lrd_bind_col(Csr1, 2, =;ID_D12, 0, 0);
lrd_exec(Csr1, 0, 0, 0, 0, 0);
lrd_fetch(Csr1, -4, 15, 0, PrintRow14, 0);
...
```

Note: You can specify a full path for the DLL. If you do not specify a path, **lr_load_library** searches for the DLL using the standard sequence used by the C++ function, LoadLibrary on Windows platforms. On Linux platforms you can set the **LD_LIBRARY_PATH** environment variable (or the platform equivalent). The **lr_load_dll** function uses the same search rules as **dlopen**. For more information, see the main pages for **dlopen** or its equivalent.

How to Load a DLL Globally

This task describes how to load a DLL globally, to make its functions available to all your Vuser scripts. Once the DLL is loaded, you can call any function defined within the DLL, without having to declare it in your script.

Load a DLL Globally

1. Add a list of the DLLs you want to load to the appropriate section of the *mdrv.dat* file, located in your application's *dat* folder.

   Use the following syntax:

   *PLATFORM*_DLLS=*my_dll1*.dll, *my_dll2*.dll, ...

   replacing the word *PLATFORM* with your specific platform. For a list of platforms, see the beginning section of the *mdrv.dat* file.

   For example, to load DLLs for Winsock Vusers on an NT platform, add the following statement to the mdrv.dat file:

   ```
   [WinSock]
   ExtPriorityType=protocol
   WINNT_EXT_LIBS=wsrun32.dll
   WIN95_EXT_LIBS=wsrun32.dll
   LINUX_EXT_LIBS=liblrs.so
   SOLARIS_EXT_LIBS=liblrs.so
   HPUX_EXT_LIBS=liblrs.sl
   AIX_EXT_LIBS=liblrs.so
   LibCfgFunc=winsock_exten_conf
   UtilityExt=lrun_api
   ExtMessageQueue=0
   ExtCmdLineOverwrite=-WinInet No
   ExtCmdLineConc=-UsingWinInet No
    WINNT_DLLS=user_dll1 .dll, user_dll2 .dll, ...
   ```

2. Call the function defined in the DLL in the appropriate place within your script.

.dat Files

There are two .dat files used by VuGen: vugen.dat and mdrv.dat.

vugen.dat

This vugen.dat file resides in the M_LROOT\dat folder and contains general information about VuGen, to be used by both the VuGen and the Controller.

```
[Templates]
RelativeDirectory=template
```

The Templates section indicates where the templates are for the VuGen protocols. The default entry indicates that they are in the relative *template* folder. Each protocol has a subfolder under *template*, which contains the template files for that protocol.

The next section is the **GlobalFiles** section.

```
[GlobalFiles]
main.c=main.c
@@TestName@@.usr=test.usr
default.cfg=test.cfg
default.usp=test.usp
```

The GlobalFiles section contains a list of files that VuGen copies to the test folder whenever you create a new test. For example, if you have a test called "user1", then VuGen will copy main.c, user1.usr and user1.cfg to the test folder.

The ActionFiles section contains the name of the file containing the Actions to be performed by the Vuser and upon which to perform iterations.

```
[ActionFiles]
@@actionFile@@=action.c
```

In addition to the settings shown above, *vugen.dat* contains settings that indicate the operating system and other compilation related settings.

mdrv.dat

The mdrv.dat file contains a separate section for each protocol defining the location of the library files and driver executables. For information about how to use this file to create a new protocol, see "How to Create a New Vuser Type" on page 820.

Creating and Running Scripts in Linux

Creating and Running Scripts in Linux - Overview

You can use VuGen on a Linux environment in the following ways:

- You can use VuGen to create Vuser scripts that run on Linux platforms. You record your application in a Windows environment and run it in Linux—recording is not supported on Linux.
- Users working in Linux-only environments can program Vuser scripts. Scripts can be

programmed in C or C++ and they must be compiled into a dynamic library.

To create a script through programming, you can use a Vuser template as a basis for a larger Vuser scripts. The template provides:

- correct program structure
- Vuser API calls
- source code and makefiles for creating a dynamic library

Programming Vuser Actions

The Vuser script files, *test*.c, *test*.usr, and *test*.cfg, can be customized for your Vuser.

You program the actual Vuser actions into the *test.c* file. This file has the required structure for a programmed Vuser script. The Vuser script contains three sections: vuser_init, Actions, and vuser_end.

Note that the template defines extern C for users of C++. This definition is required for all C++ users, to make sure that none of the exported functions are modified inadvertently.

```
#include "lrun.h"
#if defined(__cplusplus) || defined(cplusplus) extern "C"
{
#endif
int LR_FUNC vuser_init(LR_PARAM p)
{
lr_message("vuser_init done\n");

return 0;
}
 int Actions(LR_PARAM p)
{
lr_message("Actions done\n");

return 0;
}
 int vuser_end(LR_PARAM p)
{
lr_message("vuser_end done\n");

return 0;
}
#if defined(__cplusplus) || defined(cplusplus)}
#endif
```

You program Vuser actions directly into the empty script, before the **lr_message** function of each section.

The *vuser_init* section is executed first, during initialization. In this section, include the connection information and the logon procedure. The *vuser_init* section is only performed once each time you run the script.

The *Actions* section is executed after the initialization. In this section, include the actual operations performed by the Vuser. You can set up the Vuser to repeat the Actions section (in the *test*.cfg file).

The *vuser_end* section is executed last, after the all of the Vuser's actions. In this section, include the clean-up and logoff procedures. The *vuser_end* section is only performed once each time you run the script.

Note: LoadRunner controls Vusers by sending SIGHUP, SIGUSR1, and SIGUSR2 Linux signals. Do not use these signals in your Vuser programs.

How to Create a Template

VuGen includes a utility that copies a template into your working folder. The utility is called mkdbtest, and is located in $M_LROOT/bin. You run the utility by typing:

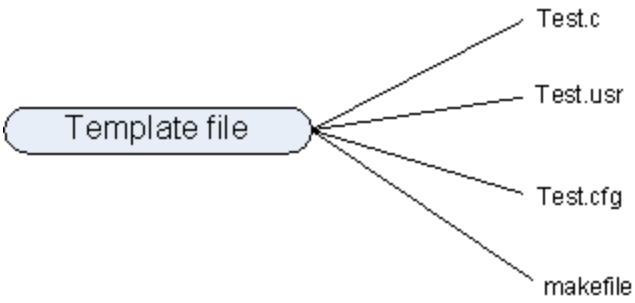
```
mkdbtest  name
```

When you run mkdbtest, it creates a folder called name, which contains the template file, name.c. For example, if you type:

```
mkdbtest test1
```

mkdbtest creates a folder called *test1*, which contains the template script, *test1.c.*

When you run the *mkdbtest* utility, a folder is created containing four files test.c, *test.usr*, *test.cfg* and *Makefile*, where *test* is the test name you specified for `mkdbtest`.



How to Configure Run-Time Settings Manually

To configure Vuser run-time settings, you modify the *default.cfg* and *default.usp* files created with the script. These run-time settings correspond to VuGen's run-time settings. (See "Run-Time Settings" on page 332.) The *default.cfg* file contains the setting for the General, Think Time, and Log options. The *default.usp* file contains the setting for the Run Logic and Pacing.

General Options

There is one General option for Linux Vuser scripts:

ContinueOnError instructs the Vuser to continue when an error occurs. To activate the option, specify 1. To disable the option, specify 0.

In the following example, the Vuser will continue on an error.

```
[General]
ContinueOnError=1
```

Think Time Options

You can set the think time options to control how the Vuser uses think time during script execution. You set the parameters Options, Factor, LimitFlag, and Limit parameters according to the following chart.

Option	Options	Factor	LimitFlag	Limit
Ignore think time	NOTHINK	N/A	N/A	N/A
Use recorded think time	RECORDED	1.000	N/A	N/A
Multiply the recorded think time by...	MULTIPLY	number	N/A	N/A
Use random percentage of recorded think time	RANDOM	range	lowest percentage	upper percentage
Limit the recorded think time to...	RECORDED/ MULTIPLY	number (for MULTIPLY)	1	value in seconds

To limit the think time used during execution, set the `LimitFlag` variable to 1 and specify the think time `Limit`, in seconds.

In the following example, the settings tell the Vuser to multiply the recorded think time by a random percentage, ranging from 50 to 150.

```
[ThinkTime]
Options=RANDOM
Factor=1
LimitFlag=0
Limit=0
ThinkTimeRandomLow=50
ThinkTimeRandomHigh=150
```

Log Options

You can set the log options to create a brief or detailed log file for the script's execution.

```
[Log]
LogOptions=LogBrief
MsgClassData=0
MsgClassParameters=0
MsgClassFull=0
```

You set the parameters LogOptions, MsGClassData, MsgClassParameters, and MsgClassFull variables according to the following chart:

Logging Type	LogOptions	MsgClassData	MsgClassParameters	MsgClassFull
Disable Logging	LogDisabled	N/A	N/A	N/A
Standard Log	LogBrief	N/A	N/A	N/A

Parameter Substitution (only)	LogExtended	0	1	0
Data Returned by Server (only)	LogExtended	1	0	0
Advanced Trace (only)	LogExtended	0	0	1
All	LogExtended	1	1	1

In the following example, the settings tell the Vuser to log all data returned by the server and the parameters used for substitution.

```
[Log]
LogOptions=LogExtended
MsgClassData=1
MsgClassParameters=1
MsgClassFull=0
```

Iterations and Run Logic

You can set the Iteration options to perform multiple iterations and control the pacing between the iterations. You can also manually set the order of the actions and their weight. To modify the run logic and iteration properties of a script, you must edit the *default.usp* file.

To instruct the Vuser to perform multiple iterations of the Actions section, set RunLogicNumOfIterations to the appropriate value.

To control the pacing between the iterations, set the RunLogicPaceType variable and its related values, according to the following chart:

Pacing	RunLogicPaceType	Related Variables
As soon as possible	ASAP	N/A
Wait between Iterations for a set time	Const	RunLogicPaceConstTime
Wait between iterations a random time	Random	RunLogicRandomPaceMin, RunLogicRandomPaceMax
Wait after each iteration a set time	ConstAfter	RunLogicPaceConstAfterTime
Wait after each iteration a random time	After	RunLogicAfterPaceMin, RunLogicAfterPaceMax

In the following example, the settings tell the Vuser to perform four iterations, while waiting a random number of seconds between iterations. The range of the random number is from 60 to 90 seconds.

```
[RunLogicRunRoot]
MercIniTreeFather=""
MercIniTreeSectionName="RunLogicRunRoot"
```

```
RunLogicRunMode="Random"
RunLogicActionOrder="Action,Action2,Action3"
RunLogicPaceType="Random"
RunLogicRandomPaceMax="90.000"
RunLogicPaceConstTime="40.000"
RunLogicObjectKind="Group"
RunLogicAfterPaceMin="50.000"
Name="Run"
RunLogicNumOfIterations="4"
RunLogicActionType="VuserRun"
RunLogicAfterPaceMax="70.000"
RunLogicRandomPaceMin="60.000"
MercIniTreeSons="Action,Action2,Action3"
RunLogicPaceConstAfterTime="30.000"
```

How to Define Transaction and Insert Rendezvous Points Manually

When programming a Vuser script without VuGen, you must manually configure the Vuser file in order to enable transactions and rendezvous. The configuration settings are listed in the test.usr file.

```
[General]
Type=any
DefaultCfg=Test.cfg
BinVuser=libtest.libsuffix
RunType=Binary
[Actions]
vuser_init=
Actions=
vuser_end=
[Transactions]
transaction1=
[Rendezvous]
Meeting=
```

Each transaction and rendezvous must be defined in the *usr* file. Add the transaction name to the Transactions section (followed by an "="). Add each rendezvous name to the Rendezvous section (followed by an "="). If the sections are not present, add them to the *usr* file as shown above.

How to Compile Scripts Manually

After you modify the template, you compile it with the appropriate *Makefile* in the script's folder. Note that for C++ compiling, you must use the native compiler (not gnu). The compiler creates a dynamic library called:

- libtest.so (solaris)
- libtest.a (AIX)
- libtest.sl (HP)

You can modify the *Makefile* and assign additional compiler flags and libraries by modifying the appropriate sections.

If you are working with a general template, you must include your application's libraries and header files. For example, if your application uses a library called testlib, include it in the LIBS section.

```
LIBS          = \
    -testlib \
    -lLrun50 \
    -lm
```

After you modify the *makefile*, type `Make` from the command line in the working folder to create the dynamic library files for the Vuser script.

After you create a script, you check it's functionality from the command line.

To run a Vuser script from the Linux command line, type:

```
mdrv -usr `pwd' test.usr
```

where *pwd* is the full path to the folder containing the Vuser script and *test*.usr is the name of the Vuser file. Check that your script communicates with the server and performs all the required tasks.

After you verify that your script is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor configuration. For more information, see the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Service Management* documentation.

Programming with the XML API

Programming with the XML API - Overview

VuGen's support for XML allows you to dynamically work with XML code and retrieve the values during test execution. Follow these steps in creating an effective XML script:

- Record a script in the desired protocol, usually Web, Web Services, or Wireless.
- Copy the XML structures into your script.
- Add XML functions from the LR API in order to retrieve dynamic data and the XML element values.

The LR API uses XPath, the XML Path language to manipulate the text in an XML document.

You can instruct VuGen to display the output values of XML elements in the Execution log window using the run-time settings. VuGen displays the line numbers, the number of matches, and the value. To allow the displaying of values, you need to enable parameter substitution. In the run-time settings, open the **General:Log** node, select **Extended log**, and select **Parameter Substitution**. For more information, see "Run-Time Settings" on page 332.

All Vuser API XML functions return the number of matches successfully found, or zero for failure.

Using XML Functions

This section provides examples of how to work with data in an XML tree. Certain functions allow you to retrieve information, and others let you write information to an XML tree. The examples use the following XML tree containing the names and extensions of several employees in the Acme organization.

```
<acme_org>
     <accounting_dept>
          <employee type='PT'>
               <name>John Smith</name>
               <extension>2145</extension>
          </employee>
     </accounting_dept>
     <engineering_dept>
          <employee type='PT'>
               <name>Sue Jones</name>
               <extension>2375</extension>
          </employee>
     </engineering_dept>
</acme_org>
```

Reading Information from an XML Tree

The functions which read information from an XML tree are:

lr_xml_extract	Extracts XML string fragments from an XML string.
lr_xml_find	Performs a query on an XML string.
lr_xml_get_values	Retrieves values of XML elements found by a query.

To retrieve a specific value through a query, you specify the tags of the parent and child nodes in a path format.

For example, to retrieve an employee name in the Accounting department, use the following string:

```
lr_xml_get_values("XML={XML_Input_Param}",
"ValueParam=OutputParam",
"Query=/acme_org/accounting_dept/employee/name",
LAST);
```

The Execution log window (with Extended logging enabled) shows the output of this function:

Output:
Action.c(20): "lr_xml_get_values" was successful, 1 match processed
Action.c(25): Query result = **John Smith**

Writing to an XML Structure

The functions which write values to an XML tree are:

lr_xml_delete	Deletes fragments from an XML string.
lr_xml_insert	Inserts a new XML fragment into an XML string.
lr_xml_replace	Replaces fragments of an XML string.
lr_xml_set_values	Sets the values of XML elements found by a query.
lr_xml_transform	Applies Extensible Stylesheet Language (XSL) transformation to XML data.

The most common *writing* function is **lr_xml_set_values** which sets the values of specified elements in an XML string. The following example uses **lr_xml_set_values** to change the phone extensions of two *employee* elements in an XML string.

First, we save the XML string to a parameter called *XML_Input_Param*. We want two values to be matched and substituted, so we prepare two new parameters, *ExtensionParam_1* and *ExtensionParam_2*, and set their values to two new phone extensions, 1111 and 2222.

lr_xml_set_values contains the argument "ValueName=ExtensionParam", which picks up the values of ExtensionParam_1 and ExtensionParam_2. The current extensions of the two employees are substituted with the values of these parameters, 1111 and 2222. The value of OutputParam is then evaluated proving that the new phone extensions were in fact substituted.

```
Action() {

     int i, NumOfValues;
     char buf[64];

     lr_save_string(xml_input, "XML_Input_Param"); // Save input as
parameter
     lr_save_string("1111", "ExtensionParam_1");
     lr_save_string("2222", "ExtensionParam_2");

     lr_xml_set_values("XML={XML_Input_Param}",
          "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
          "SelectAll=yes", "Query=//extension", LAST);

     NumOfValues= lr_xml_get_values("XML={NewXmlParam}",
          "ValueParam=OutputParam", "Query=//extension",
          "SelectAll=yes", LAST);

     for (i = 0; i < NumOfValues; i++) {/* Print the multiple values
of MultiParam */

          sprintf(buf, "Retrieved value %d : {OutputParam_%d}", i+1,
```

```
i+1);
            lr_output_message(lr_eval_string(buf));
      }

      return 0;
}
```

Output:

```
Action.c(40): Retrieved value 1: 1111
Action.c(40): Retrieved value 2: 2222
```

Specifying XML Function Parameters

Most XML API functions require that you specify the **XML element** and a **query**. You can also indicate if you want to retrieve all results or a single one.

Defining the XML Element

For defining the XML element to query, you can specify a literal string of the XML element, or a parameter that contains the XML. The following example shows the XML input string defined as a literal string:

"XML=<employee>JohnSmith</employee>"

Alternatively, the **XML** string can be a parameter containing the XML data. For example:

"XML={EmployeeNameParam}"

Querying an XML Tree

Suppose you want to find a value within an XML tag, for example, an employee's extension. You formulate a query for the desired value. The query indicates the location of the element and which element you want to retrieve or set. The path that you specify limits the scope of the search to a specific tag. You can also search for all elements of a specific type under all nodes below the root.

For a specific path, use "`Query=/`*full_xml_path_name*`/`*element_name*"

For the same element name under all nodes, use "`Query=//`*element_name*"

In the VuGen implementation of XML functions, the scope of a query is the entire XML tree. The tree information is sent to the Vuser API functions as the value of the *xml* argument.

Multiple Query Matching

When you perform a query on an XML element, by default VuGen returns only the first match. To retrieve multiple values from a query, you specify the "`SelectAll=yes`" attribute within your functions. VuGen adds a suffix of *_index* to indicate multiple parameters. For example, if you defined a parameter by the name *EmployeeName*, VuGen creates *EmployeeName_1*, *EmployeeName_2*, *EmployeeName_3*, and so on.

lr_xml_set_values("XML={XML_Input_Param}",
"ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
"SelectAll=yes", "Query=//extension", LAST);

With functions that *write* to a parameter, the values written to the parameter can then be evaluated. For example, the following code retrieves and prints multiple matches of a query:

NumOfValues = lr_xml_get_values("Xml={XmlParam}", "Query=//name",
"**SelectAll=yes**", "ValueParam=EmployeeName", LAST);

For functions that *read* from parameters, the values of the parameters must be pre-defined. The parameter must also use the convention *ParamName_IndexNumber*, for example *Param*_1, *Param*_2, *Param*_3, and so on. This collection of parameters is also known as a parameter set.

In the following example, lr_xml_set_values reads values from the parameter set and then uses those values in the XPath query. The parameter set that represents the employee extensions, is called ExtensionParam. It has two members: ExtensionParam_1 and ExtensionParam_2. The **lr_xml_set_values** function queries the XML input string and sets the value of the first match to 1111 and the second match to 2222.

```
lr_save_string("1111", "ExtensionParam_1");
lr_save_string("2222", "ExtensionParam_2");

lr_xml_set_values("XML={XML_Input_Param}",
    "ResultParam=NewXmlParam", "ValueParam=ExtensionParam",
    "SelectAll=yes", "Query=//extension", LAST);
```

XML Attributes

VuGen contains support for attributes. You can use a simple expression to manipulate attributes of XML elements and nodes, just as you can manipulate the elements themselves. You can modify the desired attribute or only attributes with specific values.

In the following example, **lr_xml_delete** deletes the first cubicle element with the name attribute.

```
lr_xml_delete(         "Xml={ParamXml}",
                          "Query=//cubicle/@name",
                         "ResultParam=Result",
                          LAST
    );
```

In the next example, **lr_xml_delete** deletes the first cubicle element with a name attribute that is equal to Paul.

```
lr_xml_delete(         "Xml={ParamXml}",
          "Query=//cubicle/@name="Paul",
         "ResultParam=Result",
          LAST
    );
```

Structuring XML Scripts

Initially, you create a new script in your preferred protocol. You can record a session in that protocol, or you may program the entire script without recording. Structure the Actions section of the script as follows:

- XML input declaration
- The Actions section

The XML input section contains the XML tree that you want to use as an input variable. You define the XML tree as a char type variable. For example:

```
char *xml_input=
"<acme_org>"
    "<employee>"
        " <name>John Smith</name>"
        "<cubicle>227</cubicle>"
        "<extension>2145</extension>"
    "</employee>"
    "<employee>"
        "<name>Sue Jones</name>"
        "<cubicle>227</cubicle>"
        "<extension>2375</extension>"
    "</employee>"
"</acme_org>";
```

The Action section contains the evaluation of the variables and queries for the element values. In the following example, the XML input string is evaluated using **lr_save_string**. The input variable is queried for employee names and extensions.

```
Action() {

    /* Save the input as a parameter.*/
    lr_save_string(xml_input, "XML_Input_Param");
    /* Query 1 - Retrieve an employee name from the specified
element.*/
    lr_xml_get_values("XML={XML_Input_Param}",
        "ValueParam=OutputParam",
        "Query=/acme_org/employee/name", LAST);

    /* Query 2 - Retrieve an extension under any path below the
root.*/
    lr_xml_get_values("XML={XML_Input_Param}",
        "ValueParam=OutputParam",
        "Query=//extension", LAST);

    return 0;
}
```

Enhancing a Recorded Session with XML

You can prepare an XML script by recording a session and then manually adding the relevant XML and Vuser API functions.

The following example illustrates how a recorded session was enhanced with Vuser API functions. Note that the only function that was recorded was **web_submit_data**, which appears in bold.

The first section contains the XML input declaration of the variable SOAPTemplate, for a SOAP message:

```
#include "as_web.h"
// SOAP message
```

```
const char*        pSoapTemplate=
  "<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
      "<soap:Body>"
          "<SendMail xmlns=\"urn:EmailIPortTypeInft-IEmailService\"/>"
      "</soap:Body>"
  "</soap:Envelope>";
```

The following section represents the actions of the user:

```
Action1()
{
// get response body
 web_reg_save_param("ParamXml", "LB=", "RB=", "Search=body", LAST);
// fetch weather by HTTP GET
web_submit_data("GetWeather",
"Action=http://glkev.net.innerhost.com/glkev_ws/
               WeatherFetcher.asmx/GetWeather",
              "Method=GET",
              "EncType=",
              "RecContentType=text/xml",
              "Referer=http://glkev.net.innerhost.com
                  /glkev_ws/WeatherFetcher.asmx?op=GetWeather",
         "Snapshot=t2.inf",
         "Mode=HTTP",
         ITEMDATA,
         "Name=zipCode", "Value=10010", ENDITEM,
         LAST);


// Get City value
  lr_xml_get_values("Xml={ParamXml}",
                "Query=City",
                "ValueParam=ParamCity",
                LAST
    );
 lr_output_message(lr_eval_string("***** City = {ParamCity} *****"));


// Get State value
 lr_xml_get_values("Xml={ParamXml}",
                "Query=State",
                "ValueParam=ParamState",
                LAST
    );
 lr_output_message(lr_eval_string("***** State ={ParamState}*****"));


// Get several values at once by using template

lr_xml_get_values_ex("Xml={ParamXml}",


"Template="
            "<Weather>"
                "<Time>{ParamTime}</Time>"
```

```
                "<Temperature>{ParamTemp}</Temperature>"
                "<Humidity>{ParamHumid}</Humidity>"
                "<Conditions>{ParamCond}</Conditions>"
             "</Weather>",
          LAST
    );


   lr_output_message(lr_eval_string(    "***** Time = {ParamTime},
                Temperature =  {ParamTemp}, "
                   "Humidity = {ParamHumid},
                Conditions = {ParamCond} *****"));


 // Generate readable forecast
    lr_save_string(lr_eval_string("\r\n\r\n*** Weather Forecast for
{ParamCity},
             {ParamState} ***\r\n"
          "\tTime: {ParamTime}\r\n"
          "\tTemperature: {ParamTemp} deg. Fahrenheit\r\n"
          "\tHumidity: {ParamHumid}\r\n"
          "\t{ParamCond} conditions expected\r\n"
          "\r\n"),
          "ParamForecast"
    );
    // Save soap template into parameter
    lr_save_string(pSoapTemplate, "ParamSoap");


// Insert request body into SOAP template
    lr_xml_insert("Xml={ParamSoap}",
         "ResultParam=ParamRequest",
         "Query=Body/SendMail",
         "position=child",
         "XmlFragment="
             "<FromAddress>taurus@merc-int.com</FromAddress>"
             "<ToAddress>support@merc-int.com</ToAddress>"
             "<ASubject>Weather Forecast</ASubject>"
             "<MsgBody/>",
      LAST
    );
//
//"<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">"
//  "<soap:Body>"
//      "<SendMail xmlns=\"urn:EmailIPortTypeInft-IEmailService\"/>"
//         "<FromAddress>taurus@merc-int.com</FromAddress>"
//         "<ToAddress>support@merc-int.com</ToAddress>"
//         "<ASubject>Weather Forecast</ASubject>"
//         "<MsgBody/>"
//      "</SendMail>"
```

```
//    "</soap:Body>"
//"</soap:Envelope>";
//
    // Insert actual forecast text
    lr_xml_set_values("Xml={ParamRequest}",
                      "ResultParam=ParamRequest",
                      "Query=Body/SendMail/MsgBody",
                      "ValueParam=ParamForecast",
                      LAST);

// Add header for SOAP
 web_add_header("SOAPAction", "urn:EmailIPortTypeInft-IEmailService");
// Get response body
 web_reg_save_param("ParamXml", "LB=", "RB=", "Search=body", LAST);
// Send forecast to recipient, using SOAP request
 web_custom_request("web_custom_request",
   "URL=http://webservices.matlus.com/scripts/emailwebservice.dll/soap
        /IEmailservice",
        "Method=POST",
        "TargetFrame=",
        "Resource=0",
        "Referer=",
        "Body={ParamRequest}",
        LAST);
    // Verify that mail was sent
    lr_xml_find("Xml={ParamXml}",
                   "Query=Body/SendMailResponse/return",
                   "Value=0",
                   LAST
    );
    return 0;
}
```

How to Use Result Parameters

Some of the **lr_xml** functions return a result parameter, such as **ResultParam**. This parameter contains the resulting XML data after the function is executed. The result parameters will be available from the parameter list in the Select or Create Parameter dialog box.

For example, for **lr_xml_insert**, ResultParam contains the complete XML data resulting from the insertion of the new XML fragment

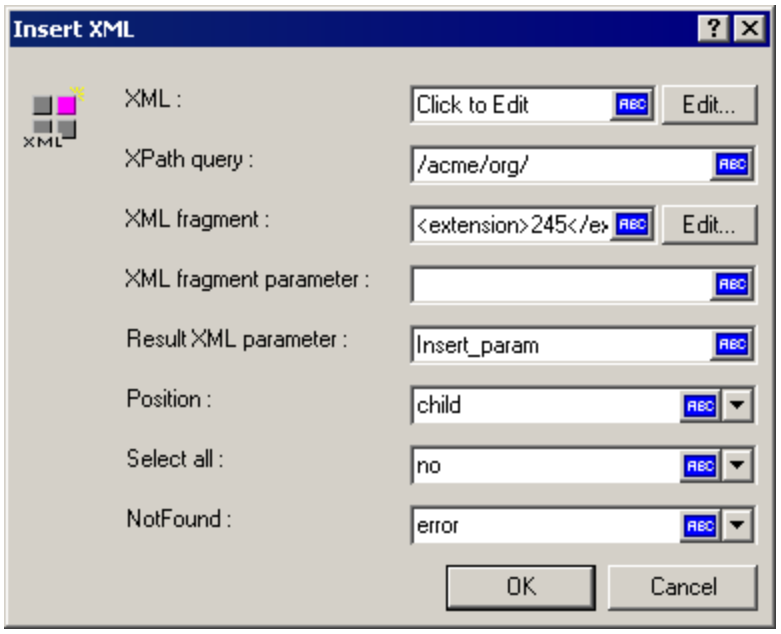
You can use the result parameters as input to other XML related functions such as Web Service calls. During replay, VuGen captures the value of the result parameter. In a later step, you can use that value as an input argument.

The functions that support result parameters are **lr_xml_insert**, **lr_xml_transform**, **lr_xml_replace**, **lr_xml_delete**, and **lr_xml_set_values**.

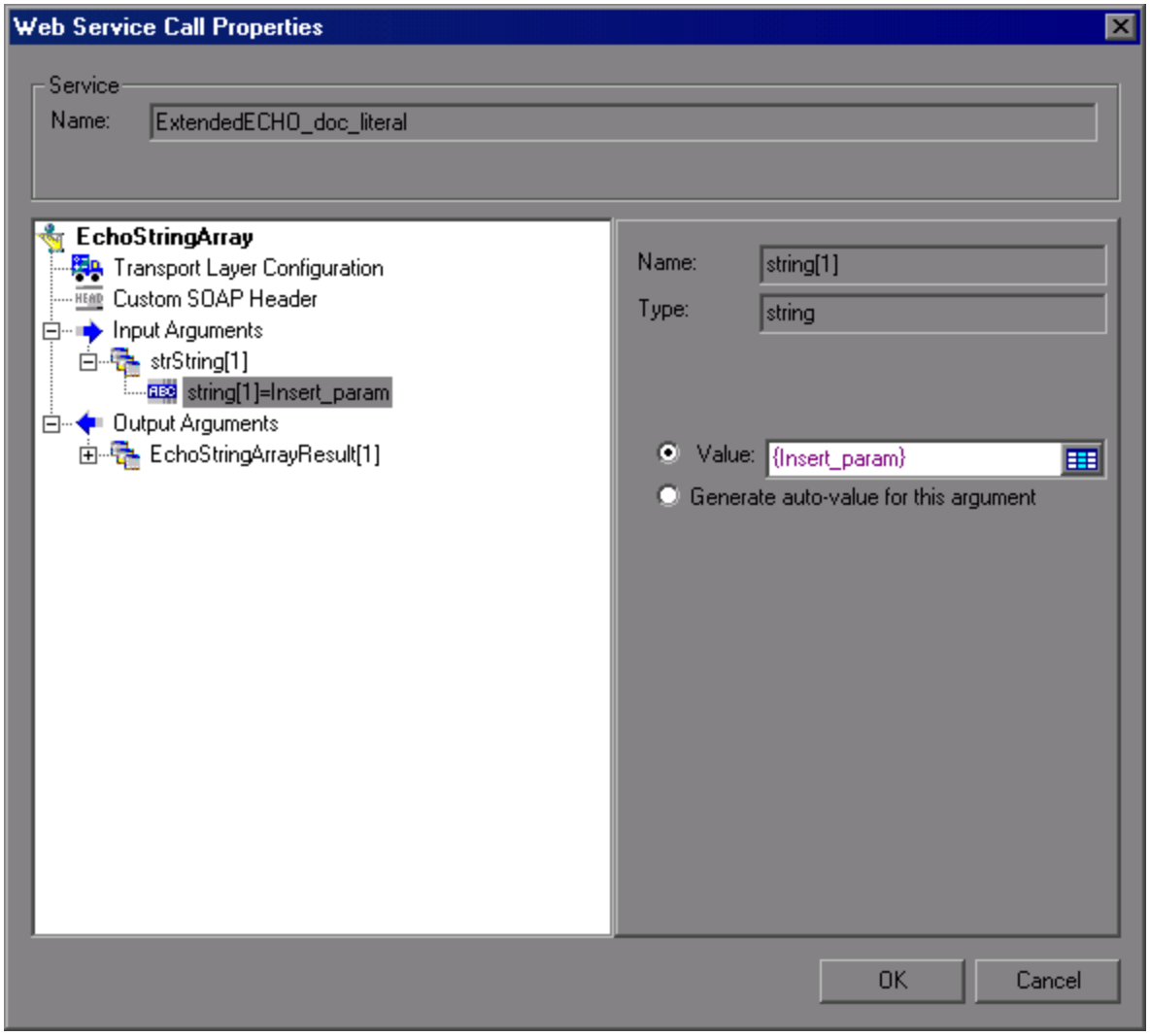
The following functions save values to a parameter other than the resultParam: **lr_xml_get_values** saves values to ValueParam and **lr_xml_extract** saves values to XMLFragmentParam. These values are also available for parameter substitution.

Use the Result Parameter as Input

1. In the **Step Navigator**, double-click on an XML step to view its Properties.
2. In the Result XML Parameter box, specify a name for the **Result XML parameter** (or ValueParam and XMLFragmentParam).



3. Reference the parameter name as in input argument.



For more information, see "New Web Service Call Dialog Box" on page 699.

HP LoadRunner Tutorial

The HP LoadRunner Tutorial is a self-paced online and printable guide, designed to lead you through the process of load testing, and familiarize you with the LoadRunner testing environment.

- To access the print version of the tutorial, click **Start > All Programs > HP Software > HP LoadRunner > Documentation > Tutorial**.

HP LoadRunner Function Reference

The HP LoadRunner Function Reference describes functions that can be used in Vuser scripts in several HP products. They can be used with supported protocols in scripts maintained in HP Virtual User Generator for use in application management and load testing. Some can be used by HP Service Test. For information about applicability, refer to the product documentation.

To access the Function Reference, click **Start > All Programs > HP Software > HP LoadRunner > Documentation > Function Reference**.

HP LoadRunner Data Format Extensions Developer Guide

HP Data Format Extensions Developer Guide assists developers in creating DFE DLLs that enable raw HTTP traffic to be converted into a maintainable, structured format that can then be maintained like a script based on a textual webpage.

To view this guide, go to **Start > All Programs > HP Software > HP LoadRunner > Documentation > Data Format Extensions Developer Guide**.


Java™
COMPATIBLE