HP Service Test

for the Windows and UNIX operating systems

Software Version: 9.00

HP Service Test User's Guide
Volume I - Using Service Test

Document Number: HPSTUG9.00-1/01
Document Release Date: June 2007
Software Release Date: June 2007

invent

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.

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

© 2000 - 2007 Mercury Interactive Corporation, All rights reserved

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

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

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

This manual's title page 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://ovweb.external.hp.com/lpe/doc_serv/

Support

Mercury Product Support

You can obtain support information for products formerly produced by Mercury as follows:

- If you work with an HP Software Services Integrator (SVI) partner (**www.hp.com/managementsoftware/svi_partner_list**), contact your SVI agent.
- If you have an active HP Software support contract, visit the HP Software Support Web site and use the Self-Solve Knowledge Search to find answers to technical questions.
- For the latest information about support processes and tools available for products formerly produced by Mercury, we encourage you to visit the Mercury Customer Support Web site at: **http://support.mercury.com**.
- If you have additional questions, contact your HP Sales Representative.

HP Software Support

You can visit the HP Software Support Web site at:

www.hp.com/go/hpsoftwaresupport/

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support 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 find more information about access levels, go to:
www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:
www.managementsoftware.hp.com/passport-registration.html

Table of Contents

PART V: APPENDIXES

Welcome to This Guide

Welcome to HP Service Test, HP's tools for creating scripts, with a focus on SOA and Web Service tests. The scripts act like virtual users, allowing you emulate real-life situations. You can use Service Test to develop a script by recording a user performing typical business processes, or you can generate automatic scripts that address your testing requirements.

Using Service Test, you can check your script's functionality using the built-in validation tools and reports. By incorporating scripts into *LoadRunner*, HP's load testing tool, you can check your service's performance under load.

Service Test integrates with *Quality Center*, HP's tool for advanced test management. Using all of Quality Center's capabilities, you can arrange your services and tests in a repository. You can create test plans, set requirements, and track defects for your Web Services.

Service Test is based on the HP Virtual User Generator, VuGen. Therefore the names *Service Test* and *VuGen* may interchange throughout this guide.

How This Guide Is Organized

This guide contains the following parts:

Part I Introducing Vuser Scripts

Introduces Vuser scripts and the various Vuser types. It also provides an overview of the steps in developing a Vuser script.

Part II Working with Service Test

Describes the Service Test interface and the recording and replaying of scripts. It also describes standard run-time settings, using data parameters and customizing a script.

Part III SOA, BPT, and Web Services Testing

This sections provide information on how to record and develop tests for SOA, Business Process Testing, and Web Services applications.

All other protocols are described in Volume II - the *Protocols user guide*.

Part IV Information for Advanced Users

Provides information for advanced users such as general debugging tips, the files generated by Service Test, and how to program scripts in Visual C and Visual Basic.

Part V Appendixes

Contains technology overviews and information about other advanced topics. Learn about calling external functions, programming in UNIX, working with foreign languages, and keyboard shortcuts.

Note: The online version of the *Service Test* user guide is a single volume, while the printed version consists of two volumes, *Volume I-Using Service Test* and *Volume II - the Protocols user guide*.

Who Should Read This Guide

This guide is for the following users of Service Test:

- Script developers
- Functional Testers
- Load Testers

This document assumes that you are moderately knowledgeable about your enterprise application.

Online Documentation

Service Test includes the following online documentation:

Readme provides last-minute news and information about Service Test and Virtual User Generator. You access the Readme from the **Start** menu.

Books Online/Printer-Friendly Documentation includes PDF versions of the documents. Click the **Help** button and choose **Books Online**.

Online Help is available from specific Service Test windows by clicking in the window and pressing **F1** or clicking the **Help** button.

Function Reference gives you online access to all of LoadRunner's functions that you can use when creating Vuser scripts, including examples of how to use the functions. Check HP's Customer Support Web site for updates to the online *Online Function Reference*.

Additional Online Resources

Web Tours sample Web site is the basis for many examples in this guide. The URL for this Web site is http://newtours.mercuryinteractive.com.

Knowledge Base. This site enables you to browse the Customer Support Knowledge Base and add your own articles. The URL for this Web site is http://support.mercury.com/cgi-bin/portal/CSO/kbBrowse.jsp.

Customer Support Web site. This site enables you to access the Knowledge Base, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is http://support.mercury.com. You can also access this Web site from the **Help** menu.

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. The URL for this Web site is www.hp.com/managementsoftware. You can also access this Web site from the **Help** menu.

Typographical Conventions

This guide uses the following typographical conventions:

UI Elements and **Function Names**	This style indicates the names of interface elements on which you perform actions, file names or paths, and other items that require emphasis. For example, “Click the **Save** button.” It also indicates method or function names. For example, "The **wait_window** statement has the following parameters:"
Arguments	This style indicates method, property, or function arguments and book titles. For example, “Refer to the *HP User’s Guide*.”
<**Replace Value**>	Angle brackets enclose a part of a file path or URL address that should be replaced with an actual value. For example, <**MyProduct installation folder**>**bin**.
Example	This style is used for examples and text that is to be typed literally. For example, “Type Hello in the edit box.”
CTRL+C	This style indicates keyboard keys. For example, “Press ENTER.”
[ ]	Square brackets enclose optional arguments.
{ }	Curly brackets indicate that one of the enclosed values must be assigned to the current argument.
...	In a line of syntax, an ellipsis indicates that more items of the same format may be included. In a programming example, an ellipsis is used to indicate lines of a program that were intentionally omitted.
|	A vertical bar indicates that one of the options separated by the bar should be selected.

Part I

Introducing Vuser Scripts

1

Developing Vuser Scripts

When testing or monitoring an environment, you need to emulate the true behavior of users on your system. HP testing tools emulate an environment in which users concurrently work on, or access your system.

To do this emulation, the human was replaced with a virtual user, or a *Vuser*. The actions that a Vuser performs are described in a *Vuser script*. The primary tool for creating Vuser scripts is HP Service Test, also known as the Virtual User Generator, *VuGen*.

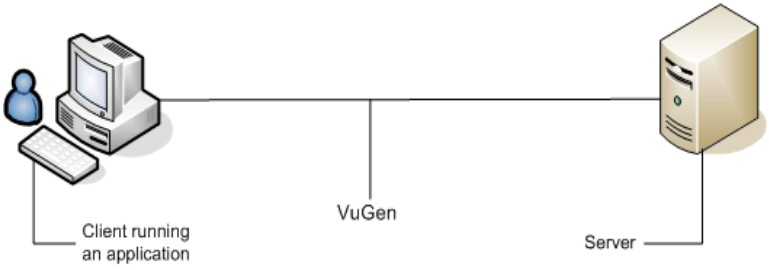
Note: The online version of the *Virtual User Generator* guide is a single volume, while the printed version consists of two volumes, *Volume I-Using Service Test* and Volume II - the *Protocols user guide*.

Introducing Vusers

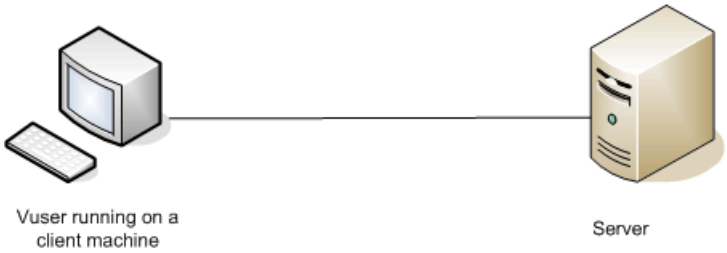
Vusers emulate the actions of human users by performing typical business processes in your application. The actions that a Vuser performs during the recording session are described in a *Vuser script*.

HP's tool for creating Vuser scripts is the Virtual User Generator, *VuGen*. You use VuGen to develop a Vuser script by recording a user performing typical business processes on a client application. VuGen records the actions that you perform during the recording session, recording only the activity between the client and the server. Instead of having to manually program the application's API function calls to the server, VuGen automatically generates functions that accurately model and emulate real world situations.

During recording VuGen monitors the client end of the database and traces all the requests sent by the user and received from the user, to the server.



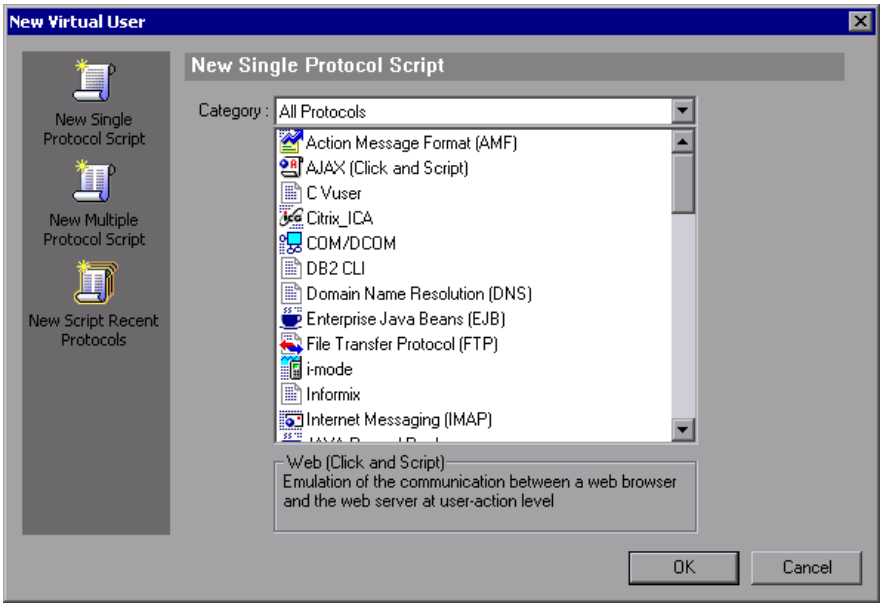
During playback, Vuser scripts communicate directly with the server by executing calls to the server API. When a Vuser communicates directly with a server, system resources are not required for the client interface. This lets you run a large number of Vusers simultaneously on a single workstation, and enables you to use only a few testing machines to emulate large server loads.



In addition, since Vuser scripts do not rely on client software, you can use Vusers to check server performance even before the user interface of the client software has been fully developed.

Using VuGen, you can run scripts as standalone tests. Running scripts from VuGen is useful for debugging as it enables you to see how a Vuser will behave and which enhancements need to be made.

VuGen enables you to record a variety of Vuser types, each suited to a particular load testing environment or topology. When you open a new test, VuGen displays a complete list of the supported protocols.



While running the Vusers, you gather information about the system's response. Afterwards, you can view this information with the Analysis tool. For example, you can observe how a server behaved when one hundred Vusers simultaneously withdrew cash from a bank's ATM.

Looking at Vuser Types

VuGen provides a variety of Vuser technologies that allow you to emulate your system. Each technology is suited to a particular architecture and results in a specific type of Vuser script. For example, you use Web Vuser Scripts to emulate users operating Web browsers. You use FTP Vusers to emulate an FTP session. The various Vuser technologies can be used alone or together, to create effective load tests or Business Process Monitor profiles.

The Vuser types are divided into the following categories:

- **All Protocols.** a list of all supported protocols in alphabetical order.
- **Application Deployment Solution.** For the Citrix and Microsoft Remote Desktop Protocol (RDP) protocols.
- **Client/Server.** For DB2 CLI, DNS, Informix, Microsoft .NET, MS SQL, ODBC, Oracle (2-tier), Sybase Ctlib, Sybase Dblib, and Windows Sockets protocols.
- **Custom.** For C templates, Java templates, Javascript, VB script, VB templates, and VBNet type scripts.
- **Distributed Components.** For COM/DCOM, and Microsoft .NET protocols.
- **E-business.** For AJAX (Click and Script), AMF, FTP, LDAP, Microsoft .NET, Web (Click and Script), Web (HTTP/HTML), and Web Services protocols.
- **Enterprise Java Beans.** For EJB Testing.
- **ERP/CRM.** For Oracle NCA, Oracle Web Applications 11i, Peoplesoft Enterprise, Peoplesoft-Tuxedo, SAP-Web, SAPGUI, SAP (Click and Script), and Siebel (Siebel-DB2CLI, Siebel-MSSQL, Siebel-Oracle, and Siebel-Web) protocols.
- **Java.** For the Java Record/Replay protocol.
- **Legacy.** For Terminal Emulation (RTE).
- **Mailing Services.** Internet Messaging (IMAP), MS Exchange (MAPI), Post Office Protocol (POP3), and Simple Mail Protocol (SMTP).
- **Middleware.** For the Tuxedo (6, 7) protocol.
- **Streaming.** For MediaPlayer (MMS) and RealPlayer protocols.

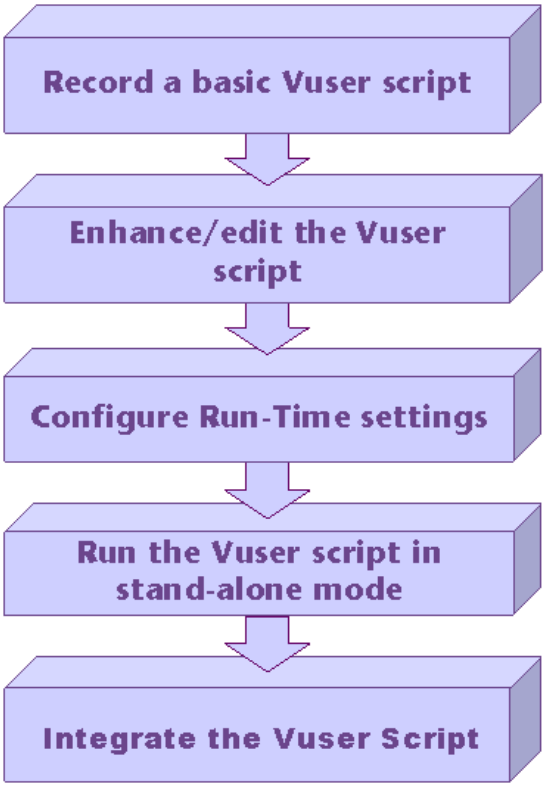
➤ **Wireless.** For Multimedia Messaging Service (MM) and WAP protocols.

To view a list of all supported protocols in alphabetical order, choose **File** > **New** and select *All Protocols* in the **Protocol Type** list box.

Note: In order to run the various protocols, you must have either a global license or licenses for the desired protocols. For more information, choose **Configuration** > **LoadRunner License** in the LoadRunner Launcher (**Start** > **Programs** > **LoadRunner** > **LoadRunner**).

The Steps of Creating Vuser Scripts

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. If you are testing Windows-based GUI applications or complex Web environments such as applets and Flash, you may need to use HP's GUI-based tools such as WinRunner and QuickTest Professional.
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, run 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. For more information, refer to the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Part II

Working with Service Test

2

Introducing Service Test

Service Test helps you develop Vuser scripts for a variety of application types and communication protocols.

The following information applies to all types of Vuser scripts except for GUI.

About Service Test

Service Test, also referred to as the Virtual User Generator, *VuGen*, is the primary tool for developing Vuser scripts.

VuGen not only records Vuser scripts, but also runs them. Running scripts from VuGen is useful for debugging. It enables you to emulate how a Vuser script will run when executed as part of a larger test.

Note: VuGen records sessions on Windows platforms only. However, a recorded Vuser script can run on both Windows and UNIX platform.

When you record a Vuser script, VuGen generates various functions that define the actions that you perform during the recording session. VuGen inserts these functions into the VuGen editor to create a basic Vuser script.

Starting VuGen

To open Service Test, choose **Start** > **Programs** > **HP Service Test** > **Service Test.**

The Service Test Start Page opens.

To open a recent script, click on it in the **Recently used scripts** list.

To open an existing script, not in the recent list, click **Open Existing Script**.

To create a new script using a recent protocol, click the protocol in the **Recently used protocols** list.

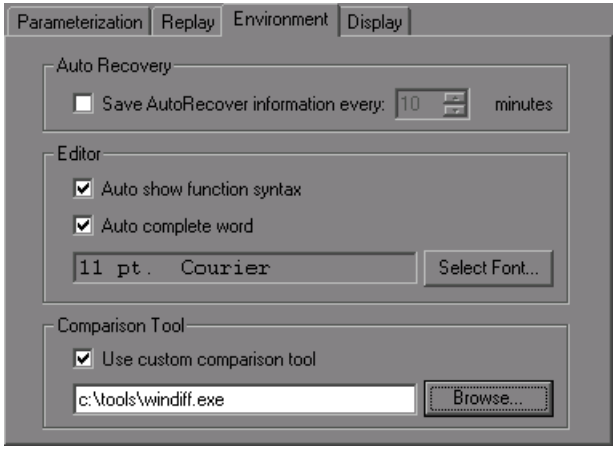
To create a new script in a protocol that is not listed, click **New Vuser Script**.

Choose **File** > **Zip Operations** > **Import From Zip File ...** to open an existing script from a zip archive.

To access Help topics for each dialog box, press F1 while clicking within the dialog box.

Understanding the VuGen Environment Options

You can set up your VuGen working environment in order to customize the auto recovery settings and the VuGen editor. You set these options from the General Options Environment tab.



Auto Recovery

The auto recovery options, allow you to restore your script's settings in the event of a crash or power outage. To allow auto recovery, select the **Save AutoRecover Information** check box and specify the time between the saves in minutes.

Editor

You can set the editor options to select a font and enable VuGen's Intellisense capabilities which automatically fill in words and function syntax.

- **Auto show function syntax.** When you type the opening parenthesis of a function, VuGen shows the syntax of the function with its arguments and prototypes. To enable the showing of the syntax globally, select the check box adjacent to this option. To disable this feature, clear the check box adjacent to the **Auto show function syntax** option. If you disable **Show Function Syntax** globally, you can still bring up the syntax by pressing Ctrl+Shift+Space or choosing **Edit** > **Show Function Syntax** after typing the opening parenthesis in the editor.
- **Auto complete word.** When you type the first underscore of a function, VuGen opens a list of functions allowing you to choose the exact function without having to manually type in the entire function. To enable word completion globally, select the check box adjacent to this option. To disable this feature, clear the check box adjacent to the **Auto complete word** option. If you disable this option globally, you can still bring up the function list box by pressing Ctrl+Space or choosing **Edit** > **Complete Word** while typing in the editor.
- **Select Font.** To set the editor font, click **Select Font**. The Font Configuration dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.

Default Environment Settings

By default, **Show Function Syntax** and **Auto complete word** are enabled globally. Auto Recovery is set to 10 seconds. To restore these values, click **Use Defaults**.

Comparison Tool

The tool to use for comparing scripts in the **Tools** > **Compare with Script** option. The default comparison tool is **WDiff**, but you can also specify a custom comparison tool.

Note: Make sure that the executable file that you specify is a valid comparison tool and that the path is correct. An executable file which is not a comparison tool will lead to unexpected results.

Setting the Environment Options

To set the environment-related options:

1. Select **Tools** > **General Options** and click the Environment tab.
2. To save the current script information for auto recovery, select the **Save AutoRecover Information** option and specify the time in minutes between the saves.
3. To set the editor font, click **Select Font**. The Font dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.
4. To use a comparison tool other than WDiff, select **Use custom comparison tool** and then specify or browse for the desired tool.
5. Click **OK** to accept the settings and close the General Options dialog box.

Viewing and Modifying Vuser Scripts

VuGen provides several views for examining the contents of your script: a text-based Script view, an icon-based Tree view with snapshots, or a icon-based Thumbnail view.

The Script and Tree views are available for most Vuser types. Many protocols also support the Thumbnail view.

Viewing the Code in Script View

The Script view lets you view the actual API functions that were recorded or inserted into the script. This view is for advanced users who want to program within the script by adding "C" or Vuser API functions as well as control flow statements.

To display the script view:

From the VuGen main menu, select **View** > **Script View**, or click the **Script** toolbar button. The script is displayed in the text-based Script view. If you are already in the Script view, the menu item is disabled.

You can expand and collapse the functions by clicking the minus or plus sign in the margin to the left of the script. This make the script neater and easier to read.

When working in Script view, you can add steps to the script using the **Insert** > **New Step** command. Alternatively, you can manually enter functions using the Complete Word and Show Function Syntax features. For more information, see "Getting Help on Vuser Functions" on page 53.

Note: If you make changes to a Vuser script while in the Script view, VuGen makes the corresponding changes in the Tree view of the Vuser script. If VuGen is unable to interpret the text-based changes that were made, it will not convert the Script view into Tree or Thumbnail view.

Viewing a Script in Tree View

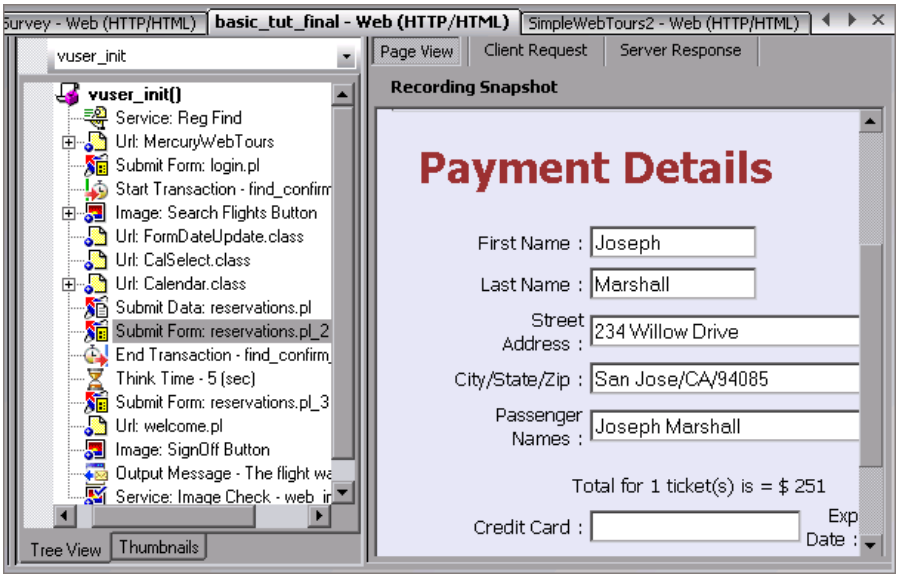
VuGen's Tree view shows the Vuser script in an icon-based format, with each step represented by a different icon.

To display the Tree view:

From VuGen's main menu, select **View** > **Tree View**, or click the **View script as tree** icon.

The Actions section of the Vuser script is displayed in the icon-based Tree view. To display a different section, choose that section in the drop-down list, above the tree.

If you are already in Tree view, the menu item is disabled.



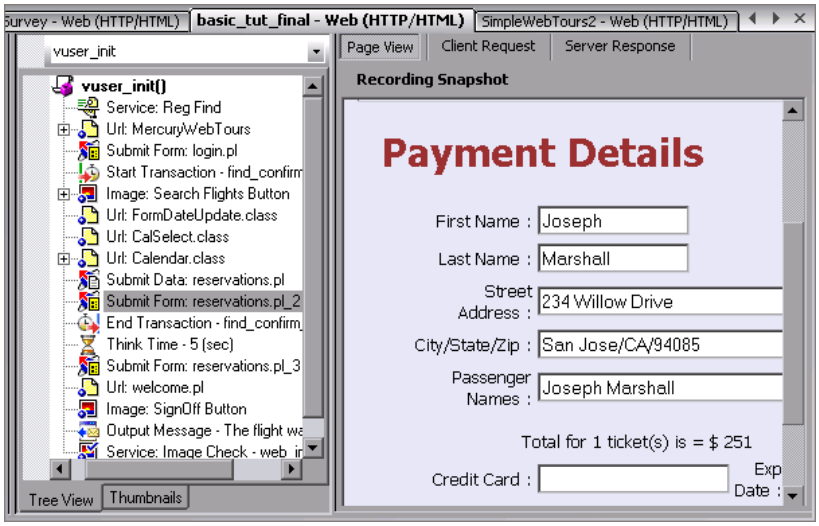
Within the Tree view, you can manipulate steps by dragging them to the desired location. You can also add additional steps between existing steps in the tree hierarchy.

To insert a step in Tree view:

1. Click on a step.
2. Choose **Insert Before** or **Insert After** from the right-click menu. The Add Step dialog box opens.
3. Choose a step and click **OK**. The Properties dialog box opens.
4. Specify the properties and click **OK**. VuGen inserts the step before or after the current step.

Understanding Snapshots

A snapshot is a graphical representation of the current step. When working in Tree view, VuGen displays the snapshot of the selected step in the right pane. The snapshot shows the client window after the step was executed.



VuGen captures a base snapshot during recording and another one during replay. You compare the Record and Replay snapshots to determine the dynamic values that need to be correlated in order to run the script. For more information, see “Correlating Vuser Scripts After Recording” in Volume II - the *Protocols user guide*.

The following toolbar buttons let you show or hide the various snapshot windows.

 Show a full window of the recorded snapshot

 Show a split window of the recorded and replayed snapshot

 Show a full window of the replayed snapshot

To view or hide snapshots:

1. Make sure you are in Tree view. If not, then switch to Tree view (**View** > **Tree View**).
2. Choose **View** > **Snapshot** > **View Snapshot**. VuGen shows the snapshot of the client window. If the snapshot is already visible, VuGen hides it.
3. Use the expanded menu of **View** > **Snapshot** to view the recorded and/or replayed snapshots. You can also use the shortcut toolbar buttons to display the desired view:

   Each time you replay the script, VuGen saves another Replay snapshot to the script's result directory: **Iteration1**, **Iteration2**, and so forth.

   By default, VuGen compares the recording snapshot to the first replay snapshot. You may, however, choose a different snapshot for comparison. To select a specific replay snapshot, choose the expanded menu of **View** > **Snapshot** > **Select Iteration**. Select a set of results and click **OK.**

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 directory, and VuGen could not save the replay snapshots.
- The step represents navigation to a resource.

Snapshot Files

Each time you replay the script, VuGen saves the snapshots in the *script* directory with an *.inf* extension. The replay snapshots are located in the script's result directory: **Iteration1**, **Iteration2**, and so forth, for each set of results.

To determine the name of the snapshot file for a Web Vuser, switch to Script view (View > Script View). In the following example, the snapshot information is represented by t1.inf.

```
web_url("WebTours",
      "URL=http://localhost/WebTours/",
      "Resource=0",
      "RecContentType=text/html",
      "Referer=",
      "Snapshot=t1.inf",
      "Mode=HTML",
      LAST);
```

For Citrix Vuser scripts, VuGen saves snapshots as bitmap files in the script's directory. To determine the name of the snapshot file, check the function's arguments in Script view.

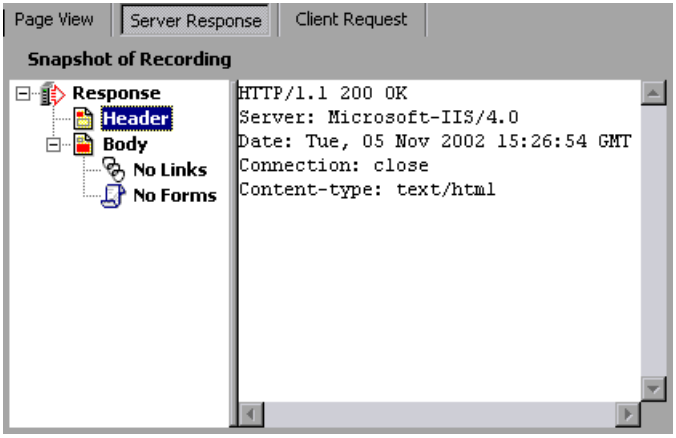
```
ctrx_sync_on_window("ICA Administrator Toolbar", ACTIVATE, 768, 0, 33,
                573, "snapshot12", CTRX_LAST);
```

Web Vuser Snapshot Tabs

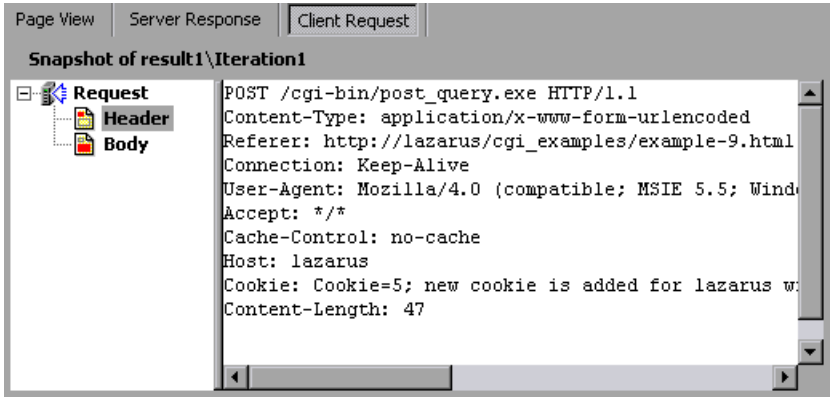
In the Snapshot window for Web Vusers, the following tabs are available:

- **Page View.** Display the snapshot in HTML as it would appear in a browser. This button is available for both the recording and replay snapshots. Use this view to make sure you are viewing the correct snapshot. In this view, however, you do not see the values that need to be correlated.
- **Server Response.** Displays the server response HTML code of the snapshot. This button is available for both the recorded and replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body with the title, links, forms, and so forth.

  To find the HTML text of an element in the source in the right pane, select the element in the left pane of the Server Response, and choose **Find Element** from the right-click menu.



➤ **Client Request.** Displays the client request HTML code of the snapshot. This button is available for both the recorded and replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body and their sub-components.



Viewing Script Thumbnails

For several Vuser types such as Web, SAPGUI and Citrix, you can view thumbnail representations of the snapshots. You can view thumbnails in either Tree view or through the Transaction Editor.

Viewing Thumbnails in Tree View

In Tree view, the **Thumbnail** tab appears at the bottom of the Tree view window.

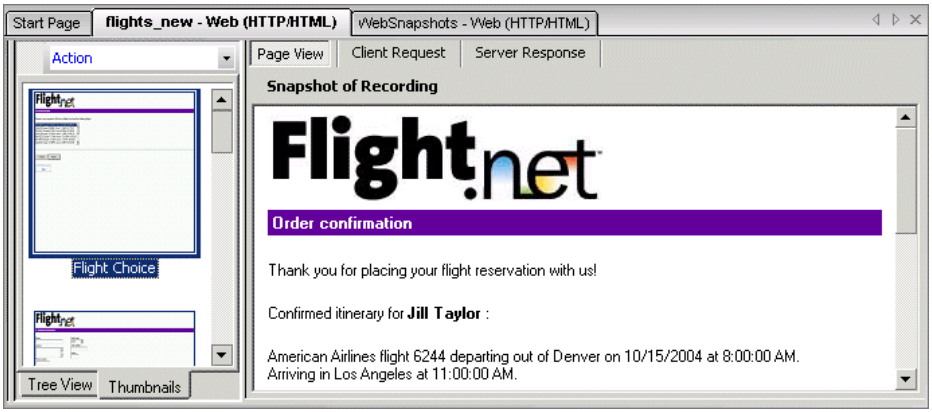
By default, the thumbnail view only shows primary steps in your script. To show all thumbnails, choose **View** > **Show All Thumbnails**. VuGen shows the thumbnails for all of the steps in the script.

Note: For multiple iterations, the VuGen shows the replay thumbnails for the last iteration. To show the thumbnails of a specific iteration, choose **View** > **Snapshot** > **Select Iteration** and select the desired iteration.

To view the thumbnails from Tree View:

1 Click the **Thumbnail** tab at the bottom of the left pane.

2 Click the desired thumbnail image to open the thumbnail's snapshot in the right pane.



3 Double-click on a thumbnail image to view a larger image. A separate window opens showing a larger view of the thumbnail.

Setting the Mode for Viewing Actions

You can instruct VuGen how to view the script by its actions (**View** > **Actions**). By default, VuGen displays the script actions in the left pane when in Script view only. You can keep the default setting (**Open Automatically**), or open the action pane in the Tree view and/or Script view.

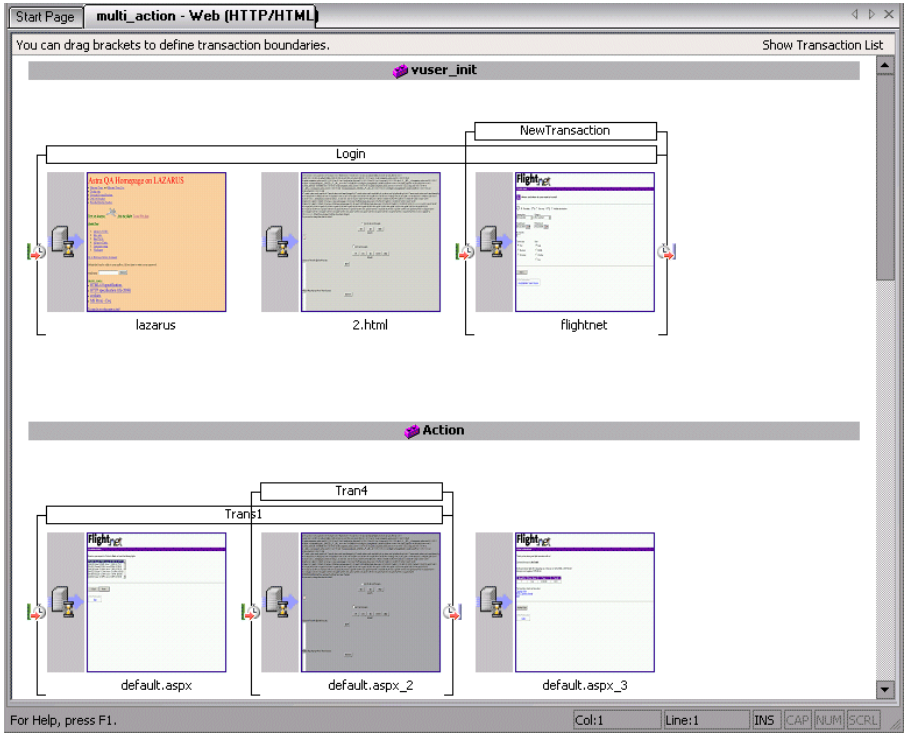
➤ **Open Automatically.** In Script view only, VuGen displays the script actions in the left pane. This is the default setting. Clear the selection to hide the actions pane.

➤ **Open in Tree Mode.** View the actions in Tree view. VuGen opens three panes: the actions, the steps, and the snapshot (if Snapshot view is enabled). You can adjust the width of each of these panes by dragging its border in the desired direction. Clear the selection to hide the actions pane.

➤ **Open in Script Mode.** View the actions in Script view. VuGen opens two panes: the actions, and the script view. You can adjust the width of the actions pane by dragging its border in the desired direction. Clear the selection to hide the actions pane.

Viewing Thumbnails in the Workflow Wizard

You can view the snapshots through the Transaction Editor. This view sorts the thumbnails by actions and provides you with an flat thumbnail view of all of the script's steps.



To view thumbnails in the Transaction Editor:

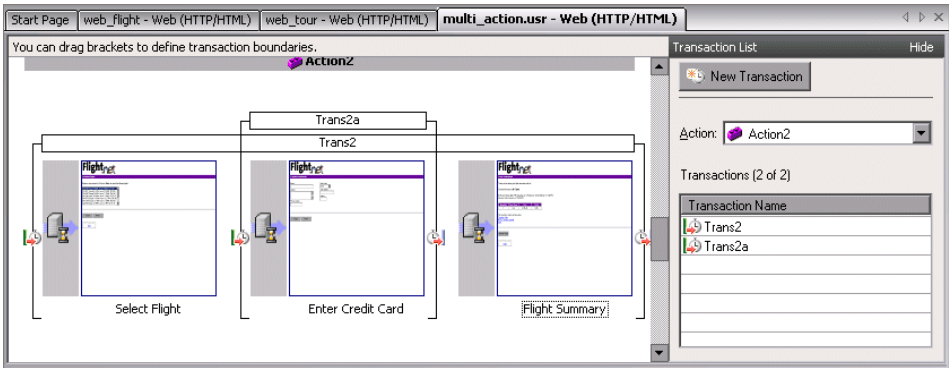
1 Click the **Tasks** button on the toolbar to open the task list pane.

2 Click the **Enhancements** > **Transactions** link. The Transaction Editor opens in the middle and right panes.

For a more encompassing view, click **Tasks** to hide the Task list.

3 In the right pane, select the action that you want to view. VuGen displays the action that you selected.

In the following example, **Action2** was selected.



Working with Thumbnails

VuGen lets you work with thumbnails by renaming them, annotating them, and viewing them in a larger size.

To view a thumbnail as a larger image:

Choose **View Larger Image** from the right-click menu or press **Alt+F6**. A separate window opens showing a larger view of the snapshot.

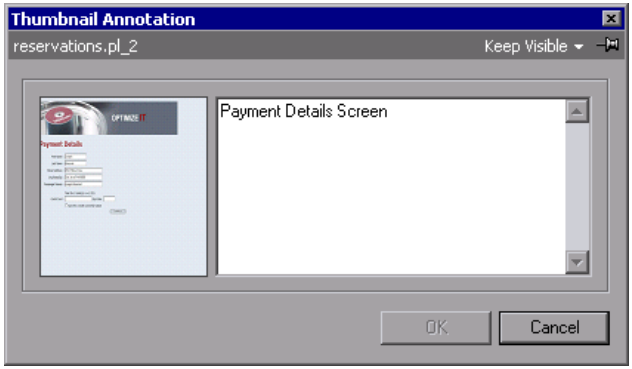
To rename a snapshot:

1 Select the snapshot and choose **Rename** from the right-click menu or press F2.

2 Type in the desired text.

To annotate a snapshot:

1 Select a snapshot and choose **Annotate** from the right-click menu or press Alt+F2. The Thumbnail Annotation dialog box opens.

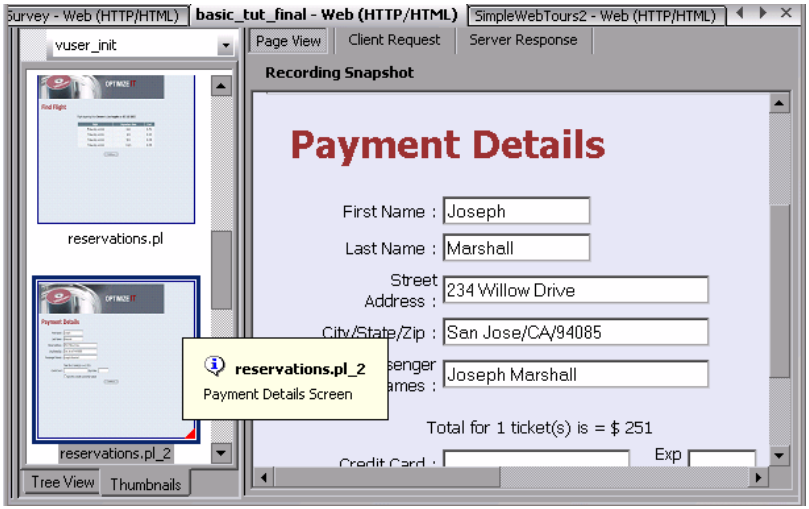


2 Type text into the right pane of the Thumbnail Annotation dialog box.

3 Click **OK** to save the annotation and close the dialog box.

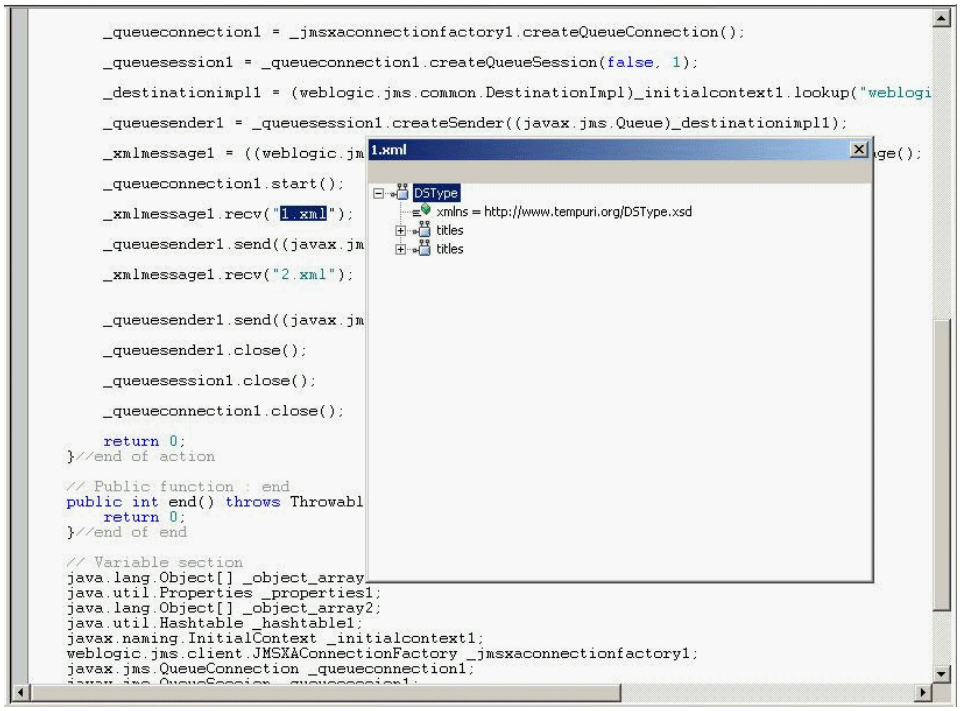
To leave the Annotation box open in order to add more text or work with other snapshots, choose **Keep Visible** from the upper right corner. The **OK** button changes to **Apply**.

After you insert an annotation for a snapshot, VuGen places a red mark in the bottom right corner of the thumbnail to indicate that an annotation exists. If you move your mouse over the thumbnail, VuGen shows a popup of the annotation text.



Using the XML Viewer

For certain protocols using XML files such as JMS, VuGen lets you view an XML structure directly from the editor window. A viewer displays the XML elements, and allows you to collapse or expand each of the nodes.



To view an file in the XML viewer:

1. In Tree view, select a step with the XML file and choose **View** > **XML**.
2. In Script view, right-click the XML file name and select **View XML**.

Running Vuser Scripts with VuGen

In order to perform testing or monitor an application with your Vuser script, you need to incorporate it into a LoadRunner scenario or Business Process Monitor profile. Before doing this, you check the script's functionality by running it from VuGen. For more information, see Chapter 15, "Running Vuser Scripts in Stand-Alone Mode."

If the script replay is successful, you can then integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, refer to the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Before you run a Vuser script, you can modify its run-time settings. These settings include the number of iterations that the Vuser performs, and the pacing and the think time that will be applied to the Vuser when the script is run. For more information on configuring run-time settings, see Chapter 13, "Configuring Run-Time Settings."

When you run a Vuser script, it is processed by an interpreter and then executed. You do not need to compile the script. If you modify a script, any syntax errors introduced into the script are noted by the interpreter. You can also call external functions from your script that can be recognized and executed by the interpreter. For more information, see Appendix A, "Calling External Functions."

Advanced users can compile a recorded script to create an executable program. For more information, see Chapter 8, "Enhancing Vuser Scripts."

Understanding VuGen Code

When you record a Vuser script, VuGen generates Vuser functions and inserts them into the script. There are two types of Vuser functions:

- General Vuser Functions
- Protocol-Specific Vuser Functions

The *general* Vuser functions and the *protocol-specific* functions together form the LoadRunner API. This API enables Vusers to communicate directly with a server. VuGen displays a list of all of the supported protocols when you create a new script. For syntax information about all of the Vuser functions, refer to the *Online Function Reference* (**Help** > **Function Reference**).

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 Chapter 8, "Enhancing Vuser Scripts" for more information.
- Send messages to the output, indicating an error or a warning.

See "Getting Help on Vuser Functions" on page 53 for a list of LR functions, and for details refer to the *Online 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 Java for Corba-Java/Rmi-Java Vusers. You can instruct VuGen to generate code in Visual Basic or Javascript. For more information, see Chapter 6, "Setting Script Generation Preferences."

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 Chapter 8, “Enhancing Vuser Scripts.” For more information about modifying a Java script, see “Programming Java Scripts” in Volume II - the *Protocols user guide*.

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.

Getting Help on Vuser Functions

You can add API Vuser functions to any script in order to enhance its capabilities. VuGen generates Vuser functions while you record. If required, you can manually insert additional functions into a script after recording. For information about typical enhancements, see Chapter 8, "Enhancing Vuser Scripts."

You can get help for VuGen's API functions in several ways:

- Online Function Reference
- Word Completion
- Show Function Syntax
- Header File

In addition, you can use the standard Search feature (**Edit** > **Find**) to locate functions within a script, or the Find In Files feature on page 258 to search all of the files in the script.

Online Function Reference

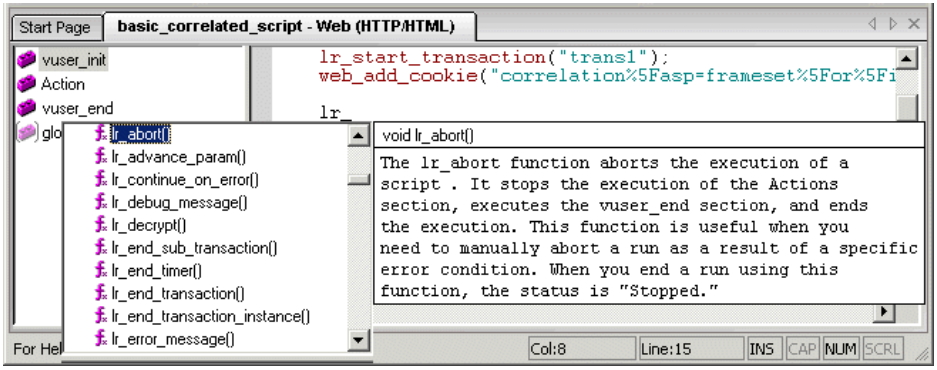
The *Online Function Reference* contains detailed syntax information about all of the VuGen functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

To open the *Online Function Reference*, choose **Help** > **Function Reference** from the VuGen interface. Then choose a protocol and select the desired category.

To obtain information about a specific function that is already in your script, place your cursor on the function in the VuGen editor, and press the F1 key.

Word Completion

As part of the *IntelliSense* enhancements, the VuGen editor incorporates the *Word Completion* feature. When you begin typing a function, after you type the first underscore, VuGen opens a list box displaying all available matches to the function prefix, along with the function's syntax and description.

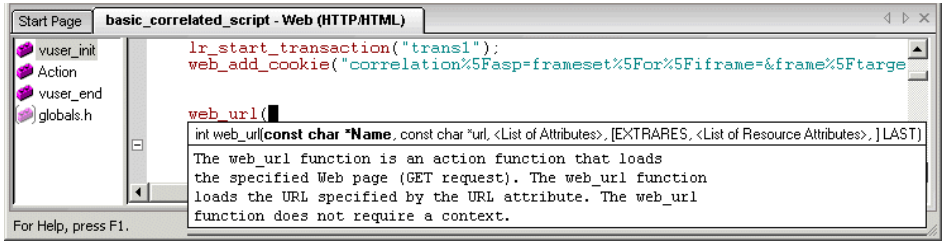


To use one of the displayed functions, select it, or scroll to the desired item and then select it. VuGen inserts the function at the location of the cursor. To close the list box, press the Esc key.

By default, VuGen uses word completion globally. To disable word completion, choose **Tools** > **General Options** and select the Environment tab. Clear the check box adjacent to the **Auto complete word** option. If you disable word completion globally, you can still bring up the list box of functions by pressing Ctrl+Space or choosing **Edit** > **Complete Word** while typing in the editor.

Show Function Syntax

An additional feature of VuGen's Intellisense, is **Show Function Syntax**. When you type the opening parenthesis of a function, VuGen shows the syntax of the function with its arguments and prototypes and a brief description.



By default, **Show Function Syntax** is enabled globally. To disable this feature, choose **Tools** > **General Options** and select the Environment tab. Clear the check box adjacent to the **Auto show function syntax** option.

If you disable **Show Function Syntax** globally, you can still bring up the syntax by pressing Ctrl+Shift+Space or choosing **Edit** > **Show Function Syntax** after typing the opening parenthesis in the editor.

Header File

All of the non-Java function prototypes are listed in the library header files. The header files are located within the *include* directory of the product installation. They include detailed syntax information and return values. They also include definitions of constants, availability, and other advanced information that may not have been included in the Function Reference.

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 & 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**
SAPGUI	**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**

3

Viewing the VuGen Workflow

VuGen's workflow screens guide you through the creation of a Vuser script that can be used for load testing or monitoring.

The following information applies to all types of Vuser scripts.

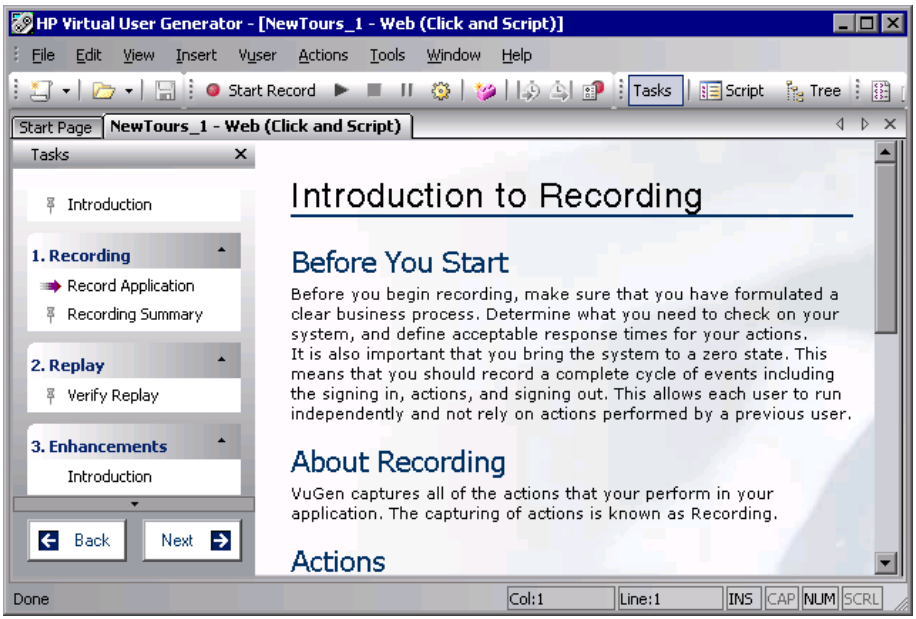
About Viewing the VuGen Workflow

VuGen's workflow screens walks you through the different steps of creating a script. First, you create a basic script, and then you adapt it for your test or production environment.

By default, after installation VuGen opens with the Workflow view. You can also work in the Tree view or Script view. The next time you start VuGen and open a script, it opens to the view that you had open when you exited VuGen. You can switch back to the wizard view by clicking on any task in the Task Pane.

Viewing the Task Pane

VuGen's left pane shows a list of the tasks required in order to create a functional script. Click on any task within the list to open that step in the wizard. VuGen indicates the current task with an arrow.



The list of tasks is divided into five parts: **Recording (Script Creation** in C and Web Services Vusers), **Verification**, **Enhancements**, **Prepare for Load**, and **Finish**.

The initial task differs slightly between Web, Web Services and non-recordable protocols, such as Custom C Vusers.

Many of the tasks have sub-tasks. The following table lists them.

Task	Sub Tasks
Recording	Record Application, Recording Summary (recordable protocols)
Script Creation	Create Script, Creation Summary (for Web Services, C))
Verification	Verify Replay
Enhancements	Introduction, Transactions, Parameterization, Content Checks (for Web Vusers)
Prepare for Load	Introduction, Iterations, Concurrent Users

Recording Steps

The Recording Section (excluding Web Services, C, and non-recordable protocols) has two steps: Recording Application and Recording Summary.

Recording Application

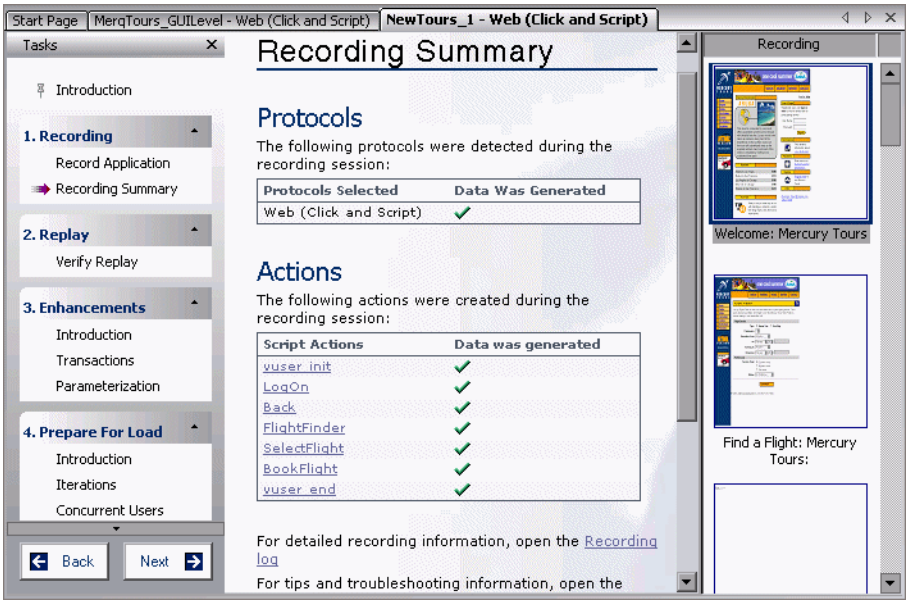
This wizard step provides an introduction to the recording process and contains the following sections.

- Before you Start
- About Recording
- Recording Process
- Recording Options
- Actions

Recording Summary

This wizard step provides a summary of the recording including the protocol information and the actions into which the session was recorded.

This step also provides thumbnails of the recorded snapshots.



Verifying the Script

The Verification section contains the **Verify Replay** step. Note that once you replay the script, you can view a Replay summary at any time by clicking **Replay Summary** in the **General** section, below the **Finish** step.

Verify Replay

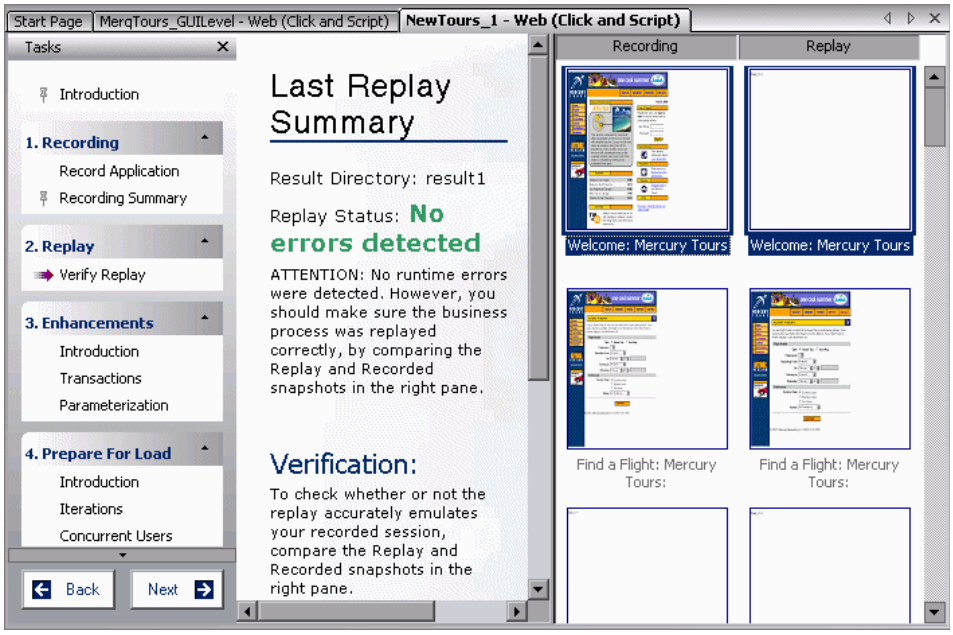
This wizard step provides an introduction to verification and contains the following sections.

- About Replay
- Run-Time Settings
- Before Replay (what to look for during replay)

Replay Summary

This wizard step provides a summary of the replay. It list the errors and provides a link to the error in the replay log.

The Replay summary also shows thumbnails of the Recording and Replay snapshots. You can visually compare the snapshots and look for discrepancies.



Note: For multiple iterations, the Replay Summary window shows the replay thumbnails for the last iteration. To show the thumbnails of a specific iteration, choose **View** > **Tree View** to switch to Tree view. Then choose **View** > **Snapshot** > **Select Iteration** and select the desired iteration.

Enhancing the Script

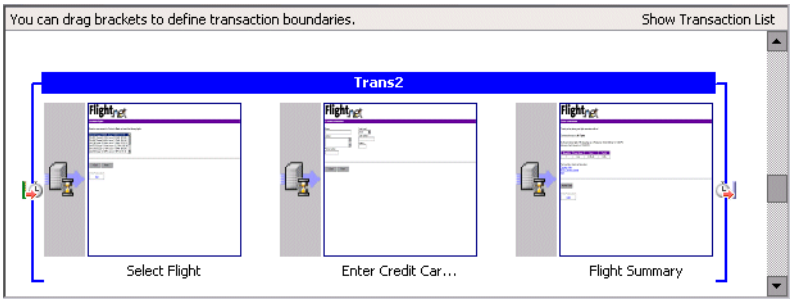
The Enhancement Section has three primary steps: Transactions, Parameterization, and Content Check.

Transactions

VuGen uses the **Transaction Editor** to allow you to add and manage transactions directly from a thumbnail view of the script.

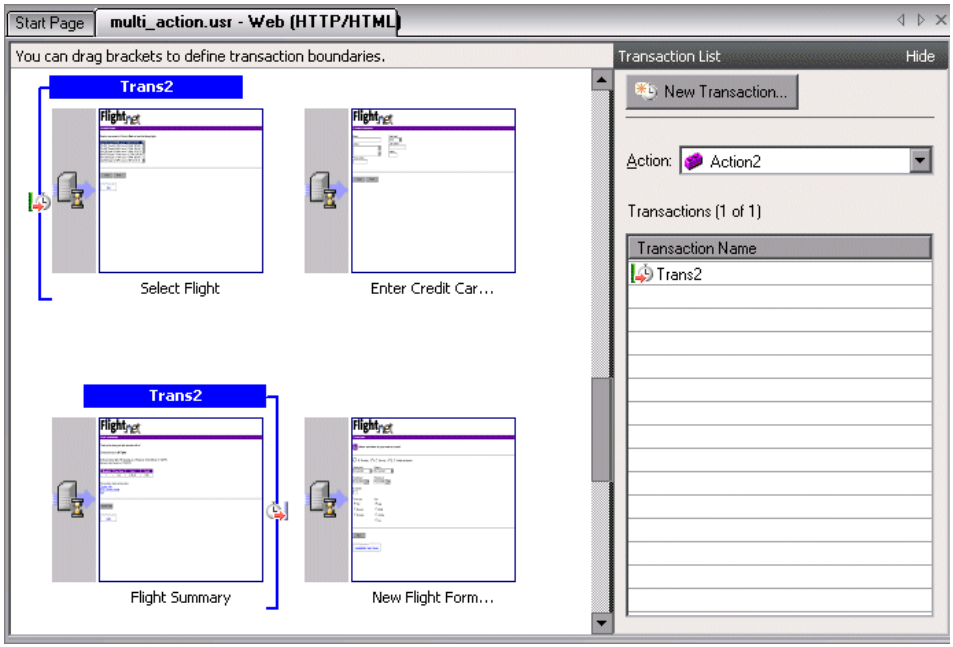
By default, VuGen only shows thumbnails for the primary steps in your script. To show thumbnails for all of the steps in your script, choose **View** > **Show All Thumbnails**.

In the following example, *Trans2* measures the time for the three steps: **Select Flight**, **Enter Credit Card**, and **Flight Summary**.

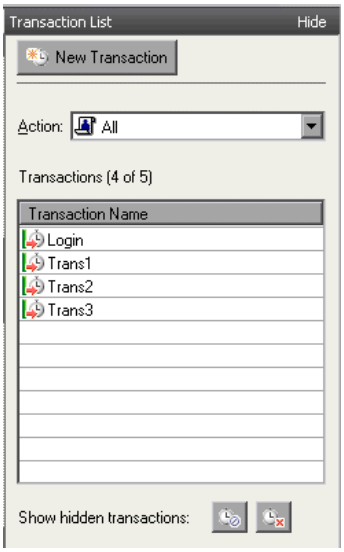


Working with the Transactions List

The transaction list, in the right pane of the Transaction editor, shows a list of the transactions in the script. You can view a complete list of the transactions in the script, or only those in a specific action.



To view all transactions, select **All** in the **Action** box. To view the transactions in a specific action, select the action name in the **Action** box.



To show and hide the Transaction list, click the **Hide** or **Show Transaction List** button in the upper right corner of the VuGen window.

By default, the transaction list only shows transactions without errors that measure the server response for primary steps in the script. It does not show:

- Non-primary steps
- Client side transactions
- Transactions with errors

Therefore, you might see the following caption above the transaction list: Transactions (2 of 4).

To show 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.

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.

To show the transactions for non-primary steps, you need to display all of the thumbnails. Choose **View** > **Show All Thumbnails**. The Transaction Editor shows the thumbnails of all the steps in the script and their transactions.

Defining Transactions in the Transaction Editor

You define a transaction in the Transaction Editor by marking the start and ending thumbnails.

To define a transaction:

1. Click **Transactions** in the Task list to open the Transaction Editor.
2. In the thumbnail area (middle pane), scroll down to the steps that you want to mark as a a transaction.

Tip: To see more thumbnails per page, click the toolbar's **Tasks** button to hide the Tasks list.

---

3. To mark a single step as a transaction, click on a thumbnail and choose **New Single-Step Transaction** from the right-click menu. VuGen prompts you to provide a name for the new transaction. If you want to expand the transactions at a later time, you can drag the transaction brackets to include additional steps.
4. To mark multiple steps as a transaction, click in the thumbnail area and choose **New Transaction** from the right-click menu or click the **New Transaction** button in the top of the right pane.
5. VuGen shows instructions in the status area above the thumbnails:

   Step 1 of 3: Select a starting point for the new transaction.

   Click the thumbnail of the transaction's first step.

Step 2 of 3: Select where the new transaction should end.

Click the thumbnail of the transaction's last step.

Step 3 of 3: Specify a name for the new transaction.

Type in a transaction name in the bracket directly above the transaction's first step.

To complete the transaction, press the Enter key.

To exit a transaction during the above sequence, press the Esc key.

6 To change the starting point of a transaction, drag the transaction opening bracket to a new location. To change the ending point of a transaction, drag the transaction closing bracket to a new location.

7 Repeat the above steps for additional transactions.

8 To rename a transaction, select its title in the right pane and choose **Rename Transaction** from the right-click menu. Type in the new name.

9 To delete a transaction, select its title in the right pane and choose **Delete Transaction** from the right-click menu.

Guidelines for the Transaction Editor

Follow these guidelines when creating and defining transactions in the Transaction Editor:

- Transactions must begin and end within a single action—they may not extend over multiple actions.
- Transaction names must be unique within your script, even between actions.
- To change the starting point of a transaction, drag the transaction opening bracket to a new location. To change the ending point of a transaction, drag the transaction closing bracket to a new location.
- Use the right-click menu to add, rename, or delete transactions.
- You can create transactions within an existing transaction. These are called *nested* transactions.

Note: If you nest transactions, close the second transaction before or at the same time as you close the first one—otherwise it won't be analyzed properly.

Parameterization

The Parameterization screen provides you with an overview of parameterizing values in your script. It guides you through the steps of parameterization:

- Locate the argument you want to parameterize
- Give the parameter a name
- Select a parameter type
- Define properties for the parameter type
- Replace the argument with a parameter

After you parameterize an argument in your script, you can return to the **Enhancements** step or replay the script.

Content Check

The Content Check wizard step lets you check your script for specific text or content.

Using a **Text Check**, you can search for a text string during replay.

Using **Content Checks**, you can instruct VuGen to search for server strings automatically using predefined rules, even if you don't know the exact text that will be returned by the server.

Prepare for Load

The fourth part of the Workflow Wizard is primarily for running load tests on your system to check the response and capacity of your machine. This part has two primary steps:

- Iterations
- Concurrent Users

Iterations

This wizard step provides an introduction to iterations and allows you to open the Run-Time settings for setting their values.

To set the number of iterations:

1. Open the Run-Time settings (F4).
2. Select the Run Logic node.
3. Specify the number of iterations.

Concurrent Users

This step guides you through the process of creating a scenario using the LoadRunner Controller.

In a scenario, you can specify the number of users to run concurrently and you can observe the behavior of your system with multiple users.

Finishing Your Script

The final step of the Workflow wizard is **Finish**.

It contains the following sections:

- **Create a Scenario.** To run a load test on your system using the LoadRunner Controller.
- **Upload to Performance Center.** To run a test through a Performance Center server installation.
- **Upload to Quality Center.** To add a test to the test repository.
- **Create Business Process Report.** Create a Microsoft Word document containing a summary of the VuGen script.

4

Recording with VuGen

VuGen creates a Vuser script by recording the communication between a client application and a server.

The following information applies to all types of Vuser scripts except for GUI.

About Recording with VuGen

VuGen creates a Vuser script by recording the actions that you perform on a client application. When you run the recorded script, the resulting Vuser emulates the user activity between the client and server.

Each Vuser script that you create contains at least three sections: *vuser_init*, one or more *Actions*, and *vuser_end*. During recording, you can select the section of the script into which VuGen will insert the recorded functions. In general, you record a login to a server into the *vuser_init* section, client activity into the *Actions* sections, and the logoff procedure into the *vuser_end* section.

After creating a test, you can save it to a zip archive and send it as an email attachment.

While recording, you can insert transactions, comments, and rendezvous points into the script. For details, see Chapter 8, "Enhancing Vuser Scripts."

Vuser 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 Chapter 13, "Configuring Run-Time Settings."

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 "Programming Java Scripts" in *VuGen Protocol user's guide*.

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

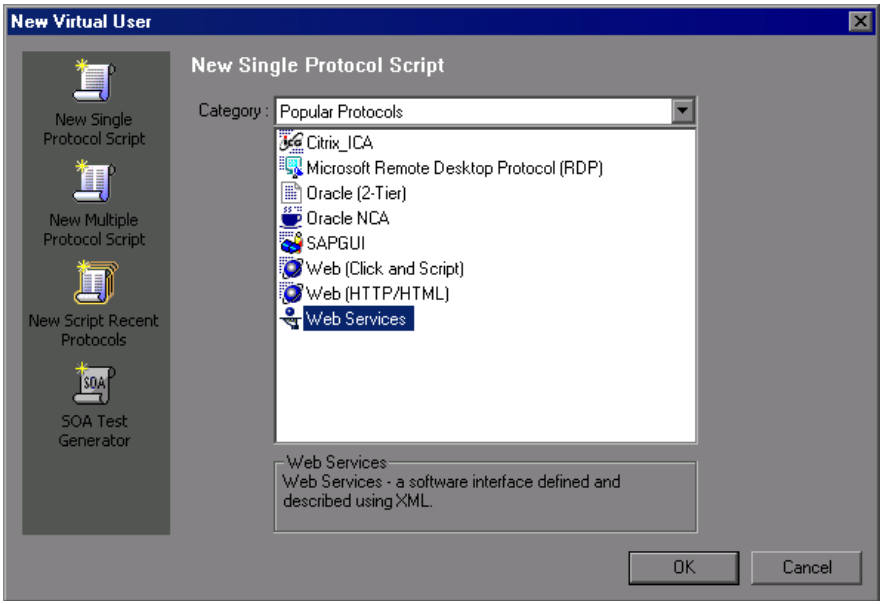
Note: Transaction Breakdown for Oracle DB is not available for actions recorded in the vuser_init section.

Creating New Virtual User Scripts

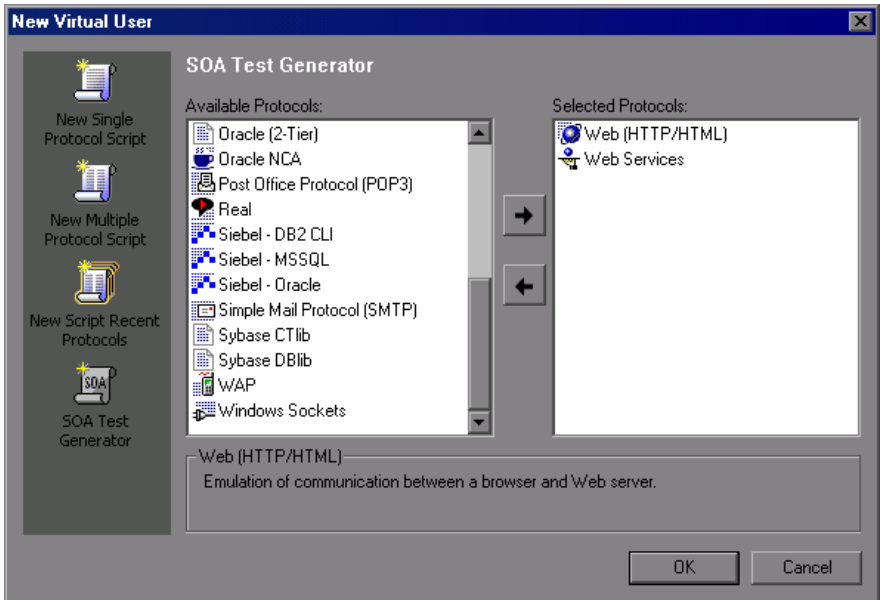
VuGen allows you to create new scripts by recording in either single or multi-protocol mode. It also provides a solution for creating scripts in a Service Oriented Architecture (SOA) environment.

The New Virtual User window opens whenever you click **New**. This dialog box provides a shortcut to the following:

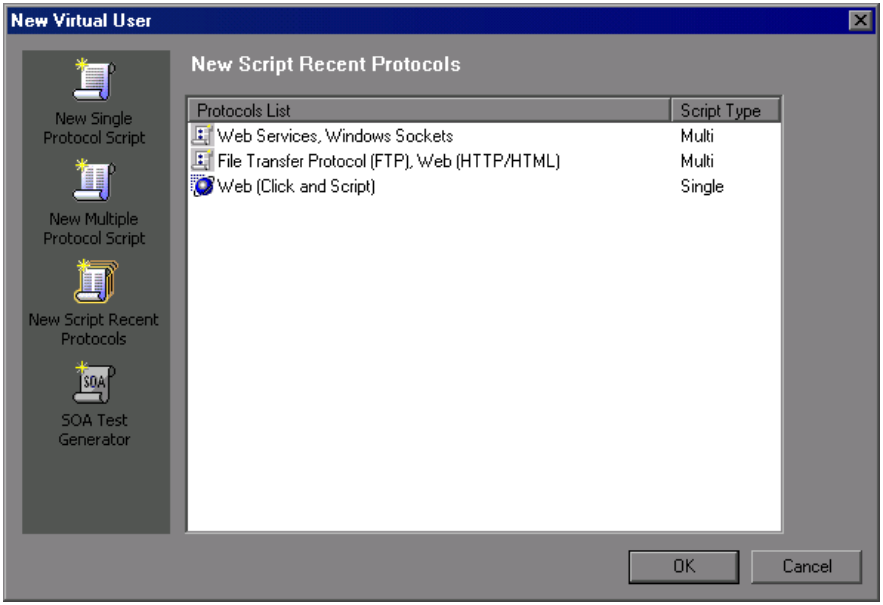
New Single Protocol Script. Creates a single protocol Vuser script. This is the default option when the Startup dialog box opens. To create a single protocol script, choose a category from the Category list (see "Choosing a Virtual User Category" on page 78), and select a protocol in the protocol list under that category.



New Multiple Protocol Script. Creates a multiple protocol Vuser script. VuGen displays all of the available protocols and allows you to specify which protocols to record. To create a multiple protocol script, choose a protocol and click the right arrow to move it into the Selected Protocols section.



New Script Recent Protocols. Lists the most recent protocols that were used to create new Vuser scripts and indicates whether they were single or multi protocol. Select a protocol from the list and click **OK** to create a new script for that protocol.



When you record a single protocol, VuGen only records 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, RealPlayer, 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, i-Mode, 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, CORBA-Java, RMI-Java, Web, WAP, i-mode, Voice XML, Oracle NCA, or RTE Vuser script, you can also record within an existing script.

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

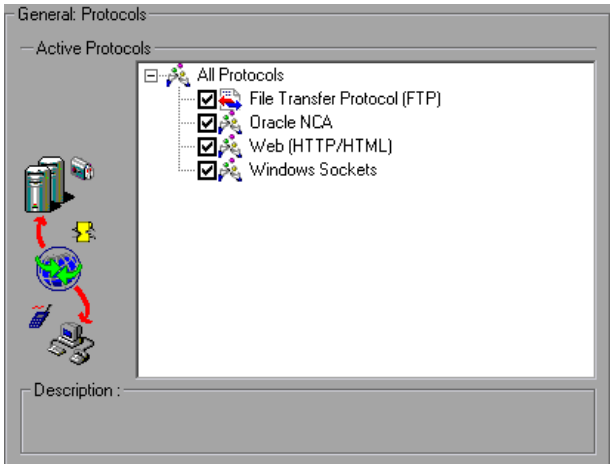
For all Java language Vusers (CORBA, RMI, Jacada, and EJB) see "Recording Java Language Vuser Scripts" in the *VuGen Protocol user's guide*.

Adding and Removing Protocols

Before recording a multi-protocol session, VuGen lets you modify the protocol list for which to generate code during the recording session. If you specified certain protocols when you created the script, you can enable or disable them using the Protocol Recording options.

To open the recording options, choose **Tools** > **Recording Options** or press Ctrl+F7. Select the **General:Protocols** node.

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.



For information on setting the Protocol options for the AMF and Web protocols, see Chapter 20, "Developing AMF Vuser Scripts" in *VuGen Protocol user's guide*.

Choosing a Virtual User Category

The Vuser types are divided into the following categories:

- **All Protocols.** a list of all supported protocols in alphabetical order
- **Application Deployment Solution:** For the Citrix and Microsoft Remote Desktop Protocol (RDP) protocols
- **Client/Server.** For DB2 CLI, DNS, Informix, Microsoft .NET, MS SQL, ODBC, Oracle 2-Tier, DB2 CLI, Sybase Ctlib, Sybase Dblib, and Windows Sockets, protocols
- **Custom.** For C Templates, Visual Basic templates, Java templates, Javascript, VBscript, and VBNet type scripts
- **Distributed Components.** For COM/DCOM and Microsoft .NET protocols
- **E-Business.** For AMF, AJAX (Click and Script), FTP, LDAP, Microsoft .NET, Web (Click and Script), Web (HTTP/HTML), and the Web Services/SOA protocols
- **Enterprise Java Beans.** For the EJB Testing protocol
- **ERP/CRM.** For Oracle NCA, Oracle Web Applications 11i, Peoplesoft Enterprise, Peoplesoft-Tuxedo, SAP-Web, SAPGUI, SAP (Click and Script), and Siebel (Siebel-DB2 CLI, Siebel-MSSQL, Siebel-Web, and Siebel-Oracle) protocols
- **Java.** For recording and replaying Java type protocols such as CORBA, RMI-Java, and JMS
- **Legacy.** For Terminal Emulation (RTE)
- **Mailing Services.** Internet Messaging (IMAP), MS Exchange (MAPI), POP3, and SMTP
- **Middleware.** For Tuxedo (6, 7) protocols
- **Streaming.** For Real and Media Player (MMS) protocols

- **Wireless.** For Multimedia Messaging Service (MMS) and WAP protocols

User-Defined Template

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.

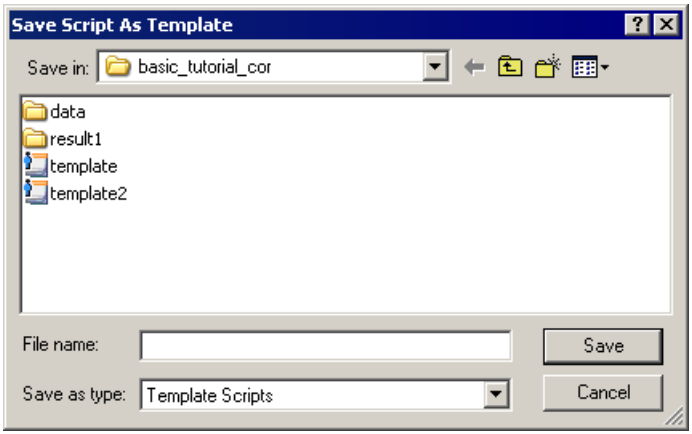
Saving and Creating Templates

You can save a script as a template or open a script from a saved template.

To save a script as a template:

1. Open the script in VuGen.

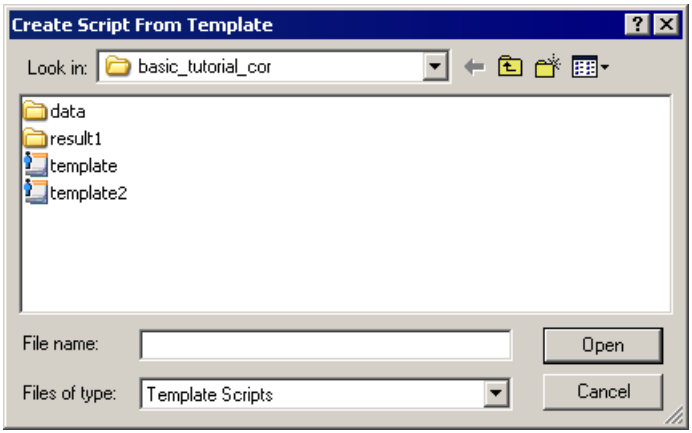
2 Select **File** > **User-Defined Template** > **Save As Template**. The Save Script As Template dialog opens. There is an icon by all existing template folders.



3 Enter the name of the template to be saved in the **File name** area and click **Save**. The script is saved as a template in the location you specified, and is added to the list.

To open a script from a template:

1 Select **File** > **User-Defined template** > **Create Script From Template**. The Create Script From Template dialog opens. There is an icon by all existing template folders.



2 Select the template you want to use as a basis for your script and click **Open**. A new script, based on the template, opens in VuGen.

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 and Script) protocol, then created a template from that script, the template can only be used with Web (Click and 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.

Creating a New Script

This section explains how to invoke VuGen and create a new script.

To create a new Vuser script:

1. Select **Start** > **Programs** > *application_name* > **Applications** > **Virtual User Generator** to start VuGen. The startup screen opens (unless you disabled it when you last opened VuGen).
2. To create a single protocol script, make a selection from the Category list and select one of the protocols.
3. To create a multi-protocol script, allowing you to record two or more protocols in a single recording session, click the **New Multiple Protocol Script** button in the left pane to enable the Protocol Selection window.

   Select the desired protocol from the **Available Protocols** list. Click the right-facing arrow to move the selection into the **Selected Protocols** list. Repeat this step for all of the desired protocols.

Note: When recording certain Oracle NCA applications, you only need to choose **Oracle NCA**—not **Web Protocol**. For details, see "Creating Oracle NCA Vuser Scripts" in *VuGen Protocol user's guide*.

---

4. To bypass this startup window the next time you open VuGen, select the **Don't show the startup dialog in the future** option. To enable it again, choose **Tools** > **General Options**, and select **Show Startup Dialog** on the Environment tab.
5. Click **OK** to close the dialog box and begin generating the Vuser script.

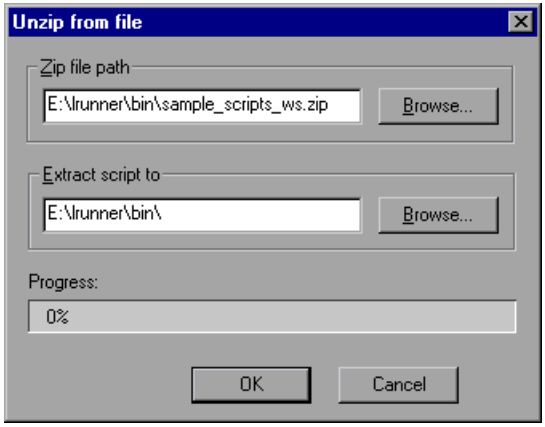
Tip: Alternatively, you can open a script from a User-Defined Template. For details, see "User-Defined Template" on page 79.

Opening an Existing Script

If you already have a script on the local machine or network, you can modify it and record additional actions.

To open an existing script:

1. To open a script stored on your local machine or a network drive, choose **File** > **Open**.
2. To open a file from a Quality Center repository (LoadRunner only), see "Opening Scripts from a Quality Center Project" on page 280.
3. To open a script stored in a compressed zip file, choose **File** > **Zip Operations** > **Import from Zip File**. After you choose a zip file, VuGen prompts you for a location at which to store the unzipped files.



4. To work from a zip file, while not expanding or saving the script files, choose **File** > **Zip Operations** > **Work from Zip File**. When you modify the script and save it, the changes are stored directly in the zip file.

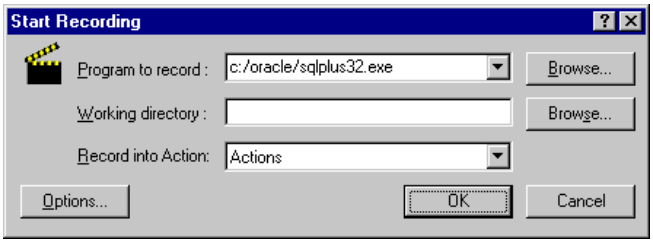
Recording Your Application

For most Vuser script types, VuGen automatically opens the Start Recording dialog box when you create the new script.

To begin recording:

1 If the Start Recording dialog box was not opened, click the **Start Recording** button. The Start Recording dialog box opens. This dialog box differs slightly for each protocol.

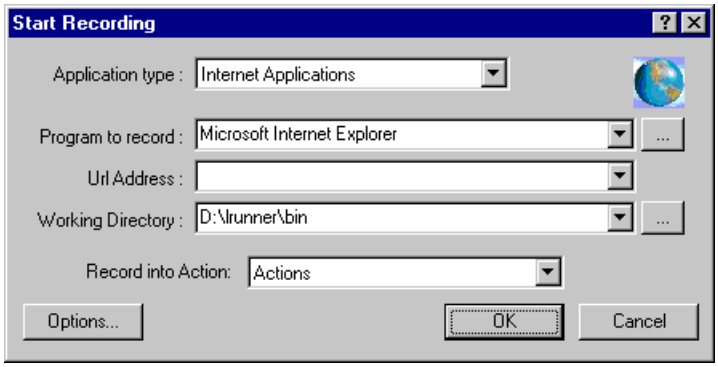
2 For most Client/Server protocols, the following dialog box opens:



Enter the program to record, the working directory, (optional) and the Action. If applicable, click **Options** to set the recording options.

3 For non-Internet applications, choose the application type: Win32 Applications or Internet Applications. For example, Web and Oracle NCA scripts record Internet Applications, while Windows Socket Vusers records a Win32 application. For Citrix ICA Vusers, VuGen automatically records the Citrix client—you only need to specify the action in **Record into Action**.

4 For Internet Applications, fill in the relevant information:

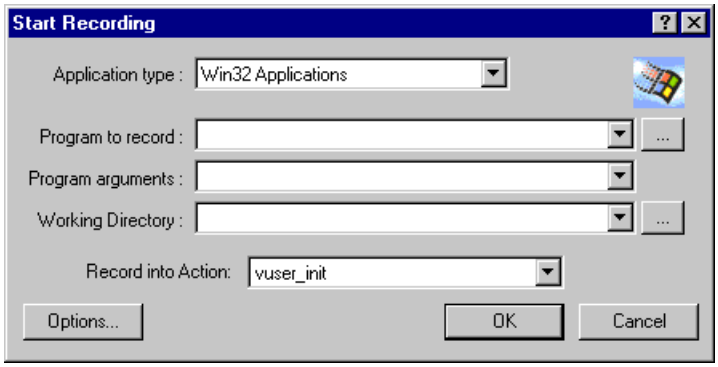


➤ **Program to record.** Select the browser or Internet application to record.

➤ **URL Address.** Specify the starting URL address.

➤ **Working Directory.** For applications that require you to specify a working directory, specify it here. The required information differs, depending on the type of Vuser script.

5 For Win32 Applications, fill in the relevant information:

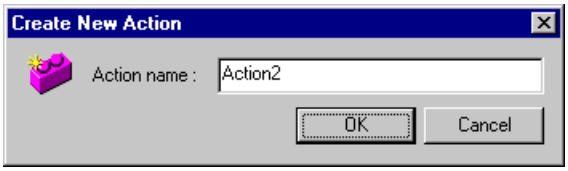


➤ **Program to record.** Enter the Win 32 application to record.

- **Program Arguments.** 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*.

- **Working Directory.** For applications that require you to specify a working directory, specify it here.

6 In the **Record into Action** box, select the section into which you want to record. Initially, the available sections are *vuser_init*, *Action*, and *vuser_end*. For single-protocol Vuser scripts that support multiple actions (Oracle NCA, Web, RTE, C Vusers, WAP, i-Mode, and VoiceXML), you can add a new section by selecting **Actions** > **Create New Action** and specify a new action name.



7 To record the application startup, select **Record the application startup** (not applicable to Java type Vuser script). To instruct VuGen not to record the application startup, clear the check box. In the following instances, it may not be advisable to record the startup:

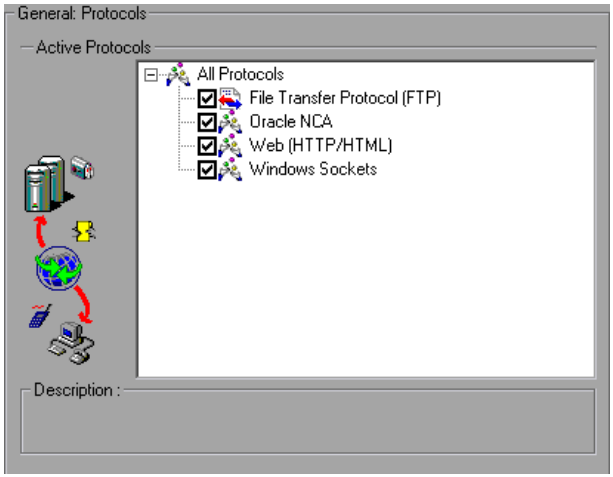
- If you are recording multiple actions, in which case you only need to perform the startup in one action.
- In cases where you want to navigate to a specific point in the application before starting to record.
- If you are recording into an existing script.

8 Click **Options** or the **Recording Options** button to open the Recording options dialog box and set the recording options. The available options may vary, depending on the recorded protocol. For more information, see their respective chapters.

9 To choose a language for code generation and to set the scripting options, click the **Script** tab. For details, see Chapter 6, “Setting Script Generation Preferences.”

10 To specify port information, click the **Port Mapping** tab. This is useful when recording SSL applications on a non-standard port. Review the list of ports. If the port you are using is not on the list, you can specify the information using the Port Mapping options. For more information, see Chapter 7, "Configuring the Port Mappings."

11 For a multi-protocol recording: To modify the list of protocols that you want to record, click the **Protocol** tab. Expand the node and select the desired protocols.



You are now ready to begin recording.

12 Click **OK** to close the dialog box and begin recording.

13 If you cleared the **Record the application startup** check box, the Recording Suspended dialog box appears. When you reach the point at which you want to start recording, click **Record**. If you decide not to record, click **Abort**.

14 VuGen starts your application and the Recording toolbar opens.

Perform typical actions within your application. VuGen simultaneously fills in the selected action section of the Vuser script. Use the floating toolbar to switch between sections during recording.

If your application or server requires authentication, VuGen will prompt you to enter a user name and password. For more information about authentication, see the appropriate section.

Ending and Saving a Recording Session

After you record a typical business process, you complete the recording session by performing the closing steps of your business process and saving the Vuser script.

To complete the recording:

1. Switch to the *vuser_end* section in the floating toolbar, and perform the log off or cleanup procedure.
2. Click the **Stop Recording** button on the Recording toolbar. The VuGen editor displays all the recorded steps, (or the recorded functions if you began in script view).
3. Click **Save** to save the recorded session. The Save Test dialog box opens (for new Vuser scripts only). Specify a script name. **Note:** Do not name the script *init*, *run* or *end*, since these names are used by VuGen.
4. To save the entire script directory as a zip file, choose **File** > **Zip Operations** > **Export to Zip File**.

   Specify which files to save. To save only runtime files, select **Runtime files** in the **Files to zip** section. By default, VuGen saves all files to the archive.

   Choose a **compression ratio**: maximum, normal, fast, super fast, or none. The greater the compression ratio, the longer VuGen will take to create the archive.

   Click **OK**.

Tip: You can save the script as a User-Defined Template. For details, see "User-Defined Template" on page 79.

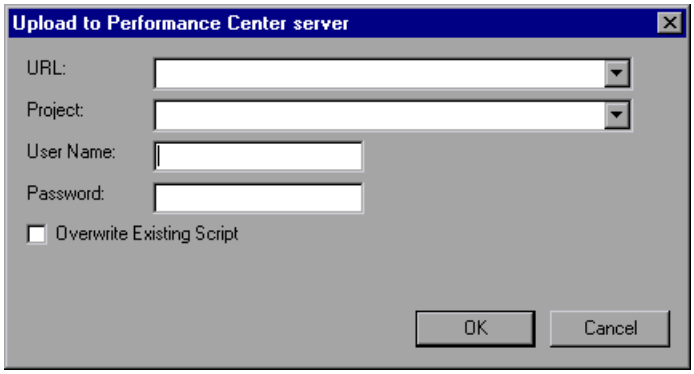
---

5 To create a zip file and send it as an email attachment, choose **File** > **Zip Operations** > **Zip and Email**.

Click **OK**. An email compose form opens.

Enter an email address and send your email.

6 For Performance Center users, you can upload your files to the repository on the server. To upload the files, choose **File** > **Upload to Performance Center server**. A dialog box opens.



- ➤ Enter the URL of the Performance Center server in the **URL** box.
- ➤ Choose a project name in the **Project** box.
- ➤ Enter your user name and password for logging on to the server.
- ➤ Click **OK** to accept the settings and close the dialog box.

Viewing the Recording Logs

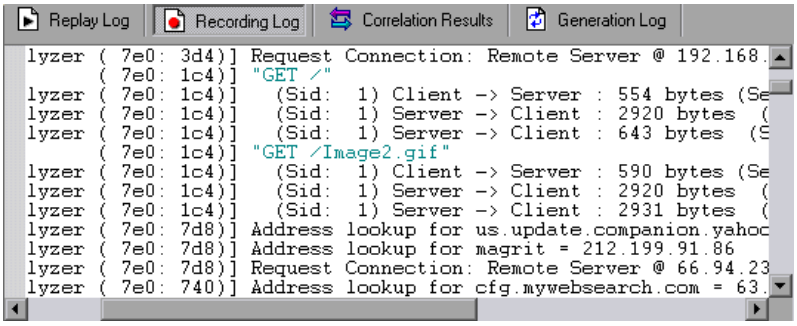
After recording, you can view the contents of the *vuser_init*, *Actions*, and *vuser_end* sections in the VuGen script editor. To display an action, select the action name in the left pane.

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 view information about the recording and the script generation by viewing the logs in the bottom window. To open the Output window, choose **View** > **Output Window** and select the Recording Log or Generation Log tabs.

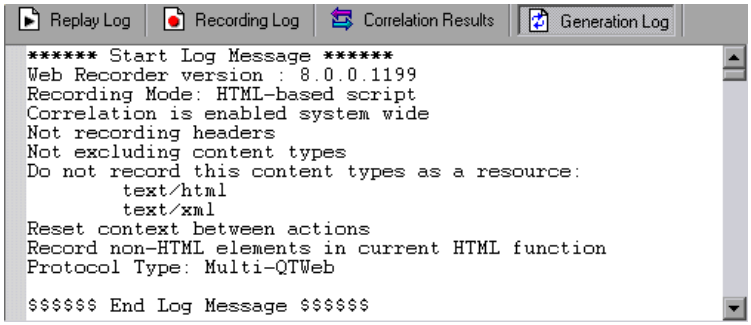
Recording Log

To view a log of the messages that were issued during recording, click the **Recording Log** tab. You can set the level of detail for this log in the **Advanced** tab of the Recording options.



Generation Log

To view a summary of the script's settings used for generating the code, select the **Generation Log** tab. This view shows the recorder version, the recording option values, and other additional information.



Using 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.

Importing from a Zip file

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

Working from a Zip file

To work from a zip file, while not expanding or saving the script files, choose **File** > **Zip Operations** > **Work from Zip File**. When you modify the script and save it, the changes are stored directly in the zip file.

Exporting to a Zip File

To save the entire script directory as a zip file, choose **File** > **Zip Operations** > **Export to Zip File**.

You can indicate whether to save all files of only runtime. By default, VuGen saves all files to the archive. To save only runtime files, select the **Runtime files** option.

You can also choose a **compression ratio**: Maximum, Normal, Fast, Super fast, or None. The greater the compression ratio, the longer VuGen takes to create the archive. The *Maximum* compression option, therefore, is the slowest.

Zip and Email

To create a zip file and send it as an email attachment, choose **File** > **Zip Operations** > **Zip and Email**. When you click **OK** in the Zip To File dialog box, VuGen compresses the file according to your settings and opens an email compose form with the zip file as an attachment.

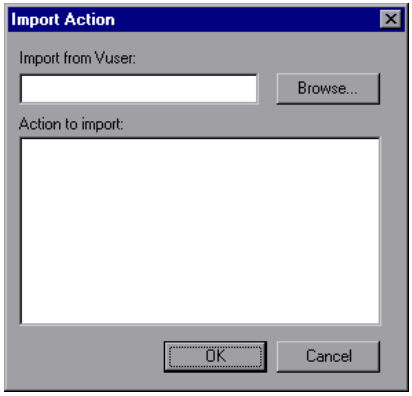
Importing Actions

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 available options are:

- **Import From Vuser.** Enter or Browse for the Vuser script from which you want to import.
- **Action to Import.** Select the Action you want to import.

To import actions into the current script:

1 Select **Actions** > **Import Action into Vuser**, or right-click the Task pane and select **Import Action into Vuser** from the right-click menu. The Import Action 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 Highlight an action and click **OK**. The action appears in your script.

4 To rearrange the order of actions, you must first enable action reordering. Right-click on any action and select **Enable Action Reorder**. Then drag the actions to the desired order. Note that when you reorder actions in the left pane of VuGen, it does not affect the order in which they are executed. To change the order of execution, use the Pacing node of the Run-Time settings as described in Chapter 13, "Configuring Run-Time Settings."

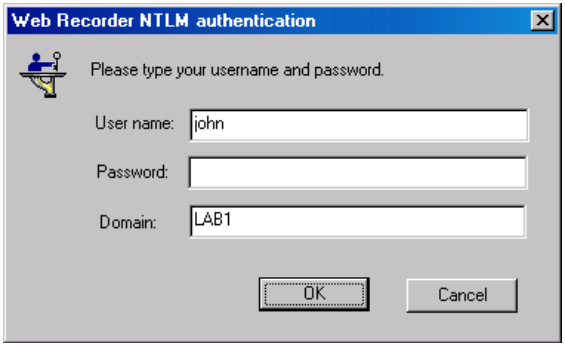
Providing Authentication Information

The following section only applies to multi-protocol recording.

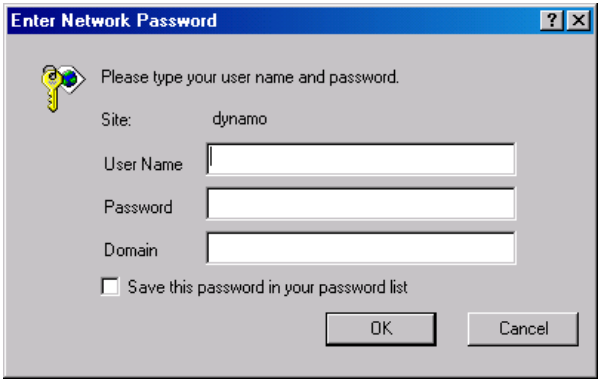
When recording a Web session that uses NTLM authentication, your server may require you to enter details such as a user name and password.

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, select it and choose **Encrypt string** from the right-click menu. 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 140.

Regenerating a Vuser Script

After recording a script, you can enhance it by adding transactions, rendezvous, messages, or comments. For more information, see Chapter 8, "Enhancing Vuser Scripts."

In addition, you can parameterize the script and correlate variables. For more information, see Chapter 9, "Working with VuGen Parameters."

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, it 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.

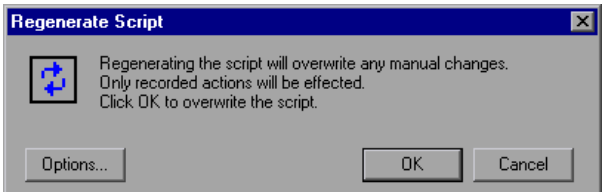
The following buttons are available from the Regenerate Script dialog box.

OK: Regenerates the Vuser script from the original Recording log. Regeneration removes all correlations and parameterizations that you performed on the script manually.

Options: When working with multi-protocol scripts, you can indicate which protocols to regenerate. To customize the regeneration, click the **Options** button in the Regenerate Vuser dialog box to open the recording options. Select the **Protocols** tab and indicate which protocols to regenerate and which to leave as is. Select the check boxes of the protocols you want to regenerate. Clear the check boxes of those you do not wish to regenerate.

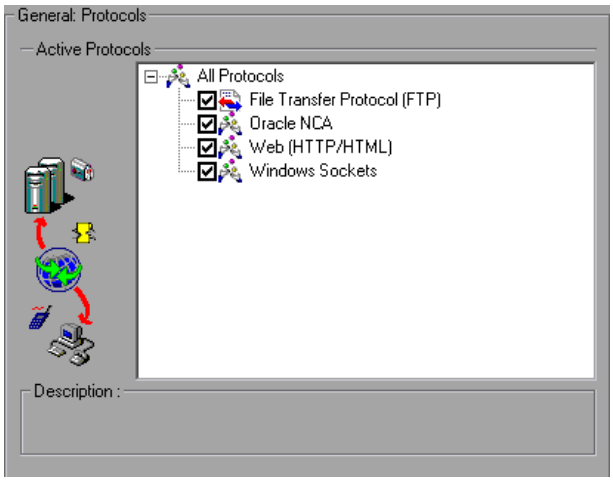
To regenerate a multi-protocol Vuser script:

1 Choose **Tools** > **Regenerate Script**. VuGen issues a warning indicating that all manual changes will be overwritten.

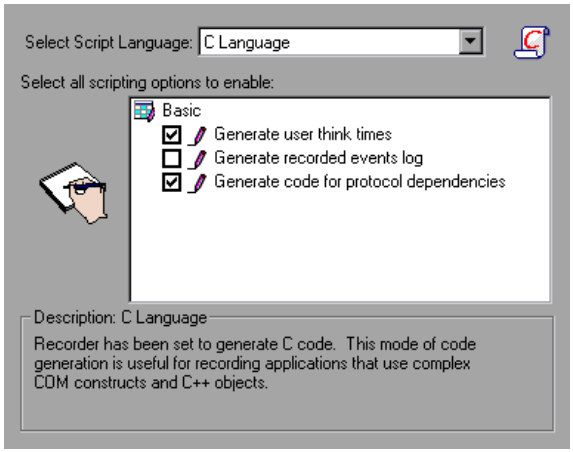


2 Click **Options** to open the Regenerate Options dialog box.

3 Select the **General:Protocols** node. Indicate which protocols to regenerate and which to leave as is. Select the check boxes of the protocols you want to regenerate. Clear the check boxes of the protocols you want to leave unchanged.



4 To change the Script options, select the **General:Script** node and select or clear the appropriate check box.



5

Creating Business Process Reports

VuGen lets you create business process reports by exporting information from your script into a Microsoft Word document.

The following information applies to the following Vuser scripts:

AJAX (Click and Script), Citrix_ICA, Oracle NCA, Oracle Web Applications 11i, PeopleSoft Enterprise, RDP, SAP (Click and Script), SAPGUI, SAP - Web, Web (Click and Script), Web (HTTP/HTML), and Web Services

About Exporting a Script to Word

At the final stage of script creation, you can create a report that will describe your business process. VuGen exports the script information to a Microsoft Word document.

You can use a pre-designed template or one provided with VuGen, to create reports with summary information about your test run.

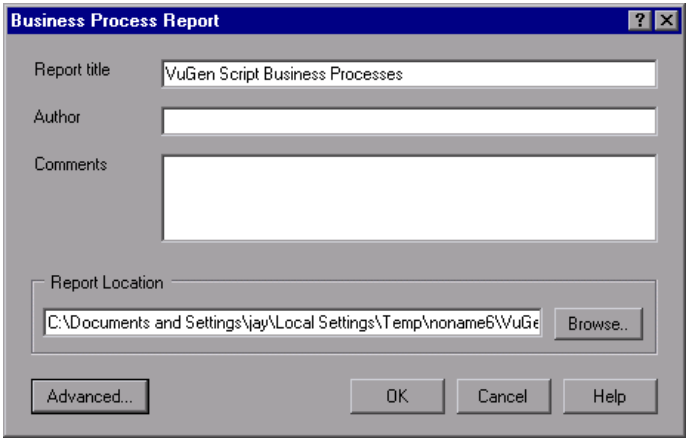
VuGen lets you customize the contents of the report by indicating what type of information you want to include.

Specifying the Report Details

Before you specify the content of the report, you give it a name and a short description. You include any relevant remarks and indicate a location at which to store the report.

To create a Business Process Report:

1 Choose **File** > **Create Business Process Report**. The Business Process Report dialog box opens.



2 Specify a title in the **Report title** box.

3 Enter your name in the **Author** box.

4 Enter any remarks in the **Comment** box.

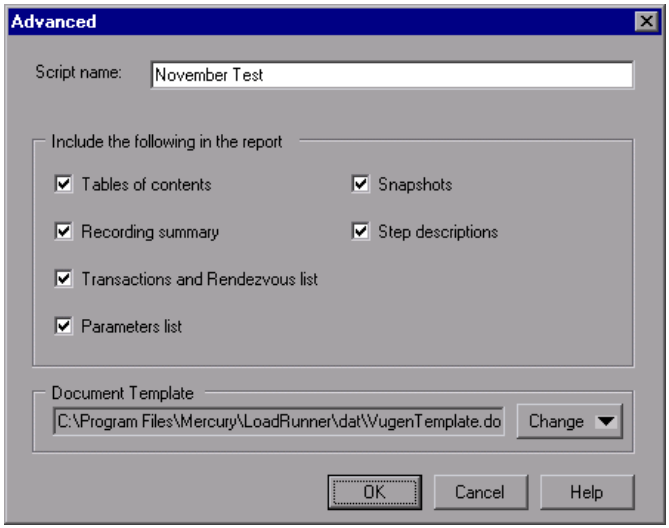
5 Accept the default report location and file name, or browse for the desired path. The default location is that of the saved script, and the default report name is **<script name>_Business_Processes.doc**

6 Click **Advanced** to specify the report's content. For more information about selecting content, see "Specifying Report Content" on page 101.

7 To generate the report, click **OK**.

Specifying Report Content

You can select content for your Business Process Report. By default all the options are enabled. By default the .usr file name appears in the **Script name** box.



You can select one or more of the following content options:

- **Table of Contents.** A table of contents indicating the page number of all of the other content in the report. If you disabled an option, it will not appear in the table of contents.
- **Recording Summary.** A summary of the recording session as it appears when you click the Recording Summary link in the Tasks list.
- **Transactions and Rendezvous list.** A comprehensive list of all of the transactions and rendezvous that were defined in the script.
- **Parameter list.** A list of all the parameters that were defined for the script. This list corresponds to the parameters listed in the Parameter List dialog box (**Vuser** > **Parameter List**).

- **Snapshots.** An actual snapshot of the recorded step, adjacent to the step name and description.

Note: Oracle NCA and Web Services reports do not include snapshots.

- **Step descriptions.** A short description of each step listed in the Tree view.
- **Document Template.** The path and file name of the template to use for the report. The default template is stored in the product's **dat** folder.

  To change the report template select **change** and specify new template with a .doc extension. If you want to create a new template, it is recommended that you use an existing template as a basis for the new one. This will ensure that the required bookmarks and styles are maintained within the new template.

6

Setting Script Generation Preferences

Before you record a script, you indicate the desired script language and the options that apply to that language.

The following information applies to all Vuser scripts that support multi-protocol recording.

About Setting Script Generation Preferences

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.

Tip: If you record a script in one language, you can regenerate it in another language after the recording. For more information, see "Regenerating a Vuser Script" on page 95.

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* options will be available. The only exception is when you record HTTP or WinSock as a single protocol script. In this case, the *Script* options are not available.

Selecting a Script Language

When you record a session, by default 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.

- **C Language.** For recording applications that use complex COM constructs and C++ objects.
- **C # Language.** For recording applications that use complex applications and environments (MS .NET protocol only).
- **Visual Basic .NET Language.** For VB .NET applications, using the full capabilities of VB.

- **Visual Basic for Applications.** For VB-based applications, using the full capabilities of VB (unlike VBScript).
- **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 or control flow statements.

The following sections describe the scripting options. For all scripts, see "Applying the Basic Options" on page 105. To set the correlation options for non-C scripts, see "Understanding the Correlation Options" on page 107.

For further instructions, see "Setting Script Recording Options" on page 108.

Applying the Basic Options

The Basic script options allow you to control the level of detail in the generated script. Some options are only available for specific languages.

- **Close all AUT processes when recording stops.** Automatically closes all of the AUT's (Application Under Test) processes when VuGen stops recording (disabled by default).
- **Declare primitives as locals**. Declares primitive value variables as local variables rather than class variables (C, C#, and .NET only, enabled by default).
- **Explicit variant declaration.** Declare variant types explicitly in order to handle *ByRef* variants (Visual Basic for Applications only, enabled by default).
- **Generate fixed think time after end transaction.** Add 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. The default is 3 seconds (disabled by default).

- **Generate recorded events log.** Generate a log of all events that took place during recording (disabled by default).

- **Generate think time greater than threshold.** Use 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 (enabled by default).

- **Insert post-invocation info.** Insert informative logging messages after each message invocation (non-C only, enabled by default).

- **Insert output parameters values.** Insert output parameter values after each call (C, C#, and .NET only, disabled by default).

- **Insert pre-invocation info.** Insert informative logging messages before each message invocation (non-C only, enabled by default).

- **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, disabled by default).

- **Reuse variables for primitive return values.** Reuse the same variables for primitives received from method calls. This overrides the **Declare primitives as locals** setting (enabled by default).

- **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 (enabled by default).

- **Use full type names.** Use the full type name when declaring a new variable (C# and .NET only, disabled by default).

- **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, enabled by default).

- **Use helpers for arrays.** Use helper functions to extract components in variant arrays (Java and VB Scripting only, disabled by default).

- **Use helpers for objects.** Use helper functions to extract object references from variants when passed as function arguments (Java and VB Scripting only, disabled by default).

For further instructions, see "Setting Script Recording Options" on page 108.

Understanding the Correlation Options

Correlation allows you to save dynamic values during test execution. These settings let you configure the extent of automatic correlation performed by VuGen while recording. All of correlation options are disabled by default. The Correlation options only apply to the non-C, such as VB Applications, VBScript, and JavaScript languages.

- **Correlate arrays.** Track and correlate arrays of all data types, such as string, structures, numbers, and so on (enabled by default).
- **Correlate large numbers.** Correlate long data types such as integers, long integers, 64-bit characters, float, and double (disabled by default).
- **Correlate simple strings.** Correlate simple, non-array strings and phrases (disabled by default).
- **Correlate small numbers.** Correlate short data types such as bytes, characters, and short integers (disabled by default).
- **Correlate structures.** Track and correlate complex structures (enabled by default).

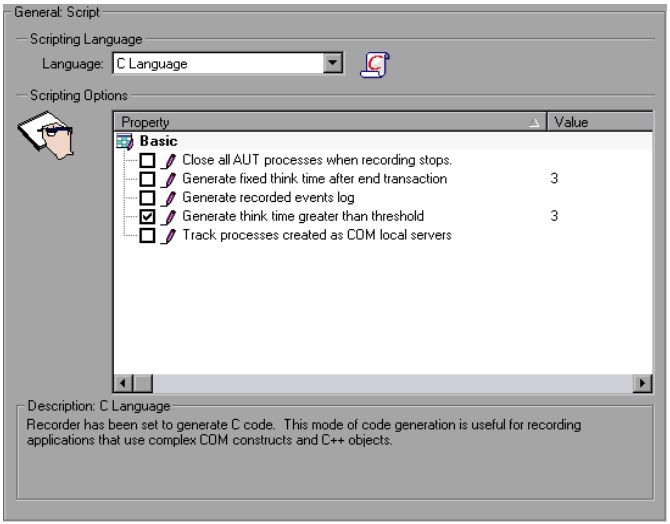
For further instructions, see "Setting Script Recording Options" on page 108.

Setting Script Recording Options

You set the Recording Options before your script related initial recording. The number of available options depends on the script generation language.

To set the script recording options:

1. Open the Recording Options. Choose **Tools** > **Recording Options** from the main menu or click **Options...** in the Start Recording dialog box. The Recording Options dialog box opens.
2. Select the **General:Script** node.



3. In the **Select Script Language** box, select a mode of code generation — *C Language* or *Visual Basic for Applications*. Use C to record applications that use complex constructs and C++ code. Use Visual Basic to record script-based applications.
4. In the **Scripting Options** section, enable the desired options by selecting the check box adjacent to it. The options are explained in the previous sections.
5. Click **OK** to save your settings and close the dialog box.

7

Configuring the Port Mappings

When working with protocols that record network traffic on a socket level, you can indicate the port to which you want to map the traffic.

The following information applies to all Vuser scripts that record on a socket level: HTTP, SMTP, POP3, IMAP, Oracle NCA, and WinSocket.

About Configuring the Port Mappings

When recording Vuser scripts that record network traffic on a socket level (HTTP, SMTP, POP3, FTP, IMAP, Oracle NCA and WinSocket), 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* options will be available. The only exception is when you record HTTP or WinSock as a single protocol script. In this case, the *Port Mapping* options are not available.

Defining Port Mappings

VuGen uses the Port Mapping settings to direct traffic via a specific server:port combination to the desired communication protocol. You can configure the Port Mapping settings in the following areas:

- **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 the WinINet DLL. Socket level sends data only if it determines that it did not originate from the WinINet dll. Port mapping applies to data that did not originate from WinINet.

- **Network-level server address mappings for.** Specifies the mappings per protocol. For example, to show only the FTP mappings, choose FTP.
- **New Entry.** Opens the Server Entry dialog box, allowing you to add a new mapping. See "Adding a New Server Entry" on page 113.
- **Edit Entry.** Opens the Server Entry dialog box, allowing you to edit the selected entry.
- **Delete Entry.** Deletes the selected entry.
- **Options.** Opens the Advanced Settings dialog box to enable auto-detection of the communication protocol and SSL level. See "Setting the Advanced Port Mapping Options" on page 115.

If you do not specify all of the port and server names, VuGen uses the following priorities in assigning data to a service:

Priority	Port	Server
1	specified	specified
2	not specified <All>	specified
3	specified	not specified <All>
4	not specified <All>	not specified <All>

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.

In addition, the following guidelines apply:

- **Port 0.** Port number 0 indicates any port.

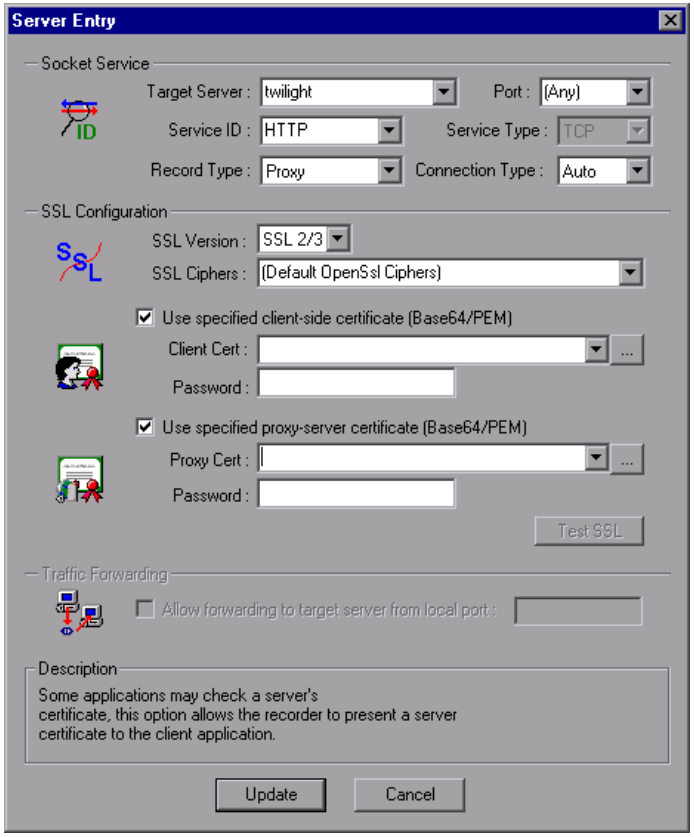
➤ **Forced mapping.** If you specify a mapping for a port number, server name, or combination server:port, VuGen forces the network traffic to use that service. For example, if you were to specify <Any> server on port 80 to use FTP, VuGen uses the FTP protocol to record that communication, even though the actual communication may be HTTP. In this instance, the Vuser script might be empty.

After you define a port mapping, it appears in the list of 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.

For further instructions, see "Setting the Port Mapping Recording Options" on page 118.

Adding a New Server Entry

You use the Server Entry dialog box to create a new entry in the list of port mappings.



Socket Service

- **Target Server.** The IP address or host name of the target server for which this entry applies. The default is All Servers.
- **Port.** The port of the target server for which this entry applies. Port 0 implies all ports.

- **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.
- **Record Type.** The type of recording—directly or through a proxy server.
- **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.

Note: SSL connections do not apply to the following Mailing Services and E-Business protocols: FTP, LDAP, SMTP, POP3, IMAP, DNS, and MAPI.

SSL Configuration

If you selected **SSL** or **auto** as the connection type, configure the relevant SSL settings in the section. These settings only apply to the new entry. You should only specify them if you have explicit information about your application's SSL encoding. Otherwise, accept the defaults.

- **SSL Version.** The preferred SSL version to use when communicating with the client application and the server. By default is SSL 2/3 is used. 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.
- **SSL Cipher.** The preferred SSL cipher to use when connecting with a remote secure server.
- **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.

Traffic Forwarding

➤ **Allow forwarding to target server from local port.** This option 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 UNIX machines, or instances where it is impossible to launch the application server through VuGen. We 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.

For example, if you were working on a UNIX 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.

For further instructions, see "Setting the Port Mapping Recording Options" on page 118.

Setting the Advanced Port Mapping Options

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, per 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.

- **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.
- **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.
- **Log Level.** Sets the logging level for the automatic socket detection: None, Standard (Default), Debug, or Advanced Debug.

When working with the above network level protocols, it is recommended 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.

For further instructions, see "Setting the Port Mapping Recording Options" on page 118.

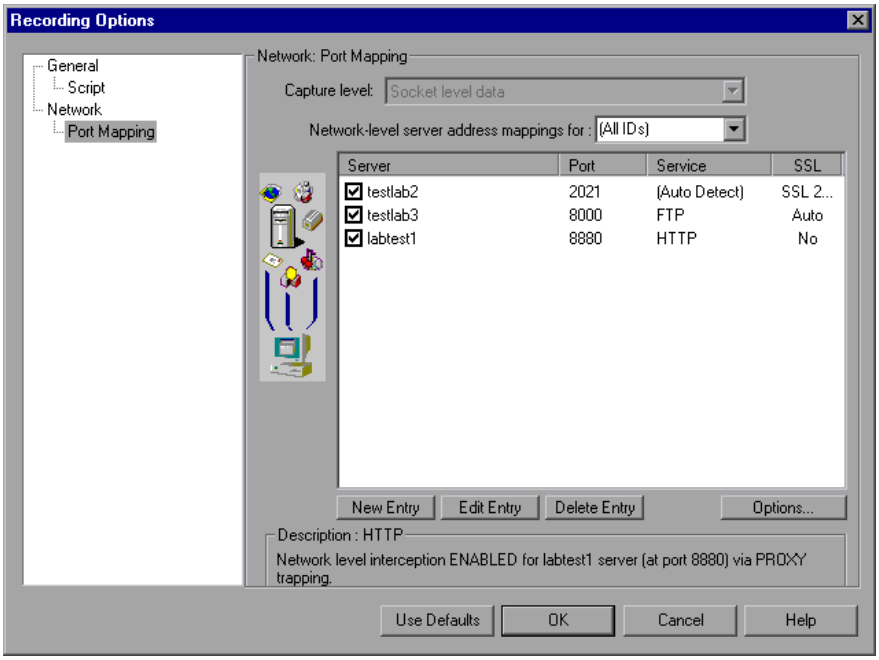
Setting the Port Mapping Recording Options

Note that you can open the Recording Options dialog box in several ways:

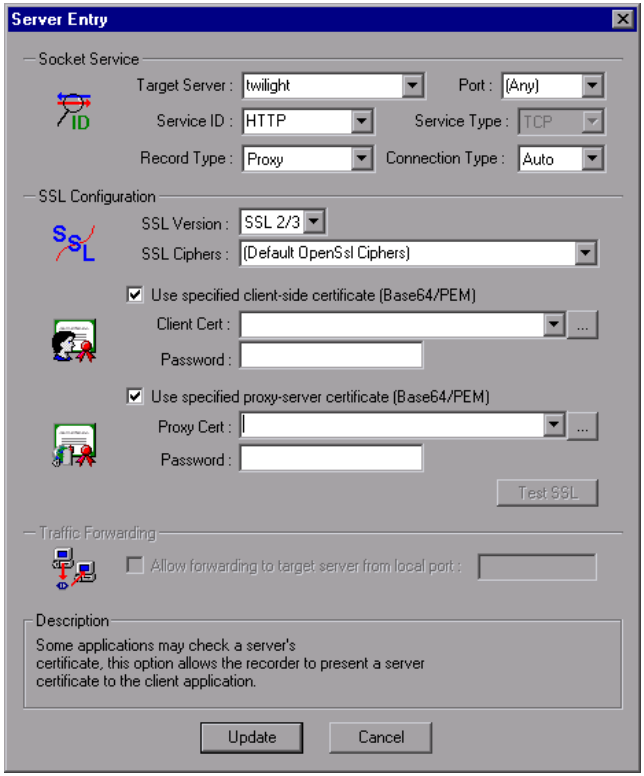
- The toolbar button: 
- The keyboard shortcut: Ctrl+F7
- The Tools menu: choose **Tools** > **Recording Options**

To set the port mapping recording options:

1 Open the Recording Options and select the **Network:Port Mapping** node.



2 To create a new server:port mapping, click **New Entry**. The Server Entry dialog box opens.



3 Enter the **Service ID, Service Type, Target Server, Target Port,** and **Connection Type** in the Socket Service section:

4 If you selected **SSL** or **auto** as the connection type, configure the relevant SSL settings in the **SSL Configuration** section. These settings only apply to the new entry. You should only specify them if you have explicit information about your application's SSL encoding. Otherwise, accept the defaults.

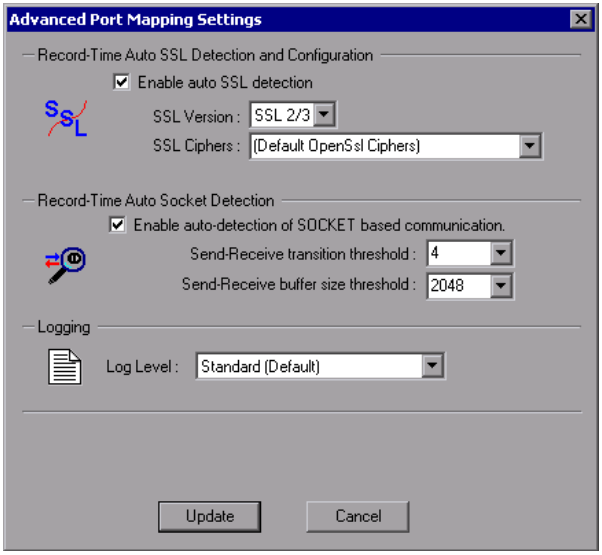
Specify the **SSL Version**, **SSL Cipher**. To use a certificate, select **Use specified client-side certificate** or **Use specified proxy-server certificate** and specify the user information.

Click **Test SSL** to check the authentication information against the server.

5 To allow traffic forwarding, select **Allow forwarding to target server from local port**, and specify a port number. Note that this option is only enabled when the **Target Server** and **Target Port** are unique (not <Any>).

6 Click **Update** to save the mapping and close the Server Entry dialog box.

7 To set automatic detection capabilities, click **Options**. The Advanced Port Mapping Setting dialog box opens.



To automatically detect SSL communication, select **Enable auto SSL detection** and specify the version and cipher information.

To automatically detect the type of communication, select **Enable auto detection of SOCKET based communication**. If required, raise the maximum number of transitions.

Select a **Log Level:** None, Standard, Debug, or Advanced Debug.

Click **Update** to accept the auto-detection options and close the dialog box.

8 To view all of the entries, select **All IDs** in the **Network-level server address mappings** box.

9 To modify an existing entry, select it and click **Edit Entry**. Note that you cannot change the server name or port number of an entry. You can only change the connection type and security settings.

10 To permanently delete a mapping, select the entry from the list and click **Delete Entry**. To temporarily disable the mapping settings for a specific entry, clear the check box adjacent to that item. To enable the mapping, select the check box.

11 Click **OK**.

8

Enhancing Vuser Scripts

You can enhance a Vuser script—either during or after recording—by adding General Vuser functions, Protocol-Specific Vuser functions, and Standard ANSI C functions.

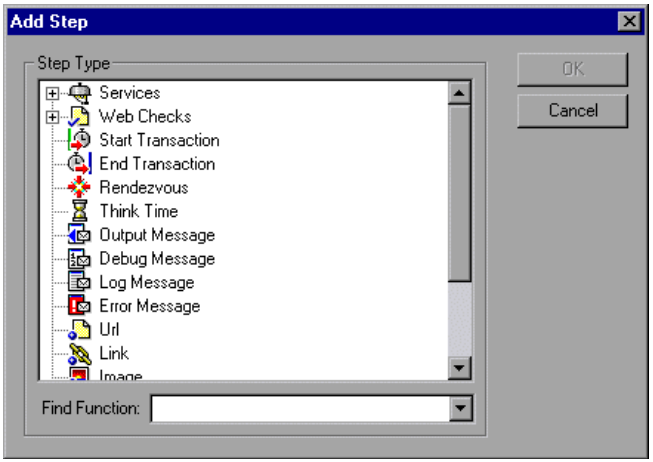
The following information applies to all Vuser Scripts.

About Enhancing Vuser Scripts

While you are recording a Vuser script, or after you record it, you can enhance its capabilities by manually adding a step, also known as a function.

To add a new step to your script.

1. Place the cursor at the desired location.
2. Choose **Insert** > **New Step**. The Add Step dialog box opens with the relevant steps for the current protocol.



3. Select a step and click **OK**. VuGen inserts the step (or function in Script view) at the location of the cursor.

The following types of functions are available from the Add Steps dialog box:

- General Vuser Functions
- Protocol-Specific Vuser Functions
- Standard ANSI C Functions

General Vuser Functions

General Vuser functions greatly enhance the functionality of any Vuser script. For example, you can use General Vuser functions to measure server performance, control server load, add debugging code, or retrieve run-time information about the Vusers participating in the test.

You can use General Vuser functions in any type of Vuser script. All General Vuser functions have an **LR** prefix. VuGen generates some General Vuser functions and inserts them into a Vuser script during recording. To use additional functions that were not automatically generated, choose **Insert** > **New Step** from VuGen's main window and select the desired function.

This chapter discusses the use of only the most common General Vuser functions. For additional information about Vuser functions, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Protocol-Specific Vuser Functions

There are several libraries of functions that you can use to enhance a Vuser script. Each library is specific to a type of Vuser. For example, you use the **LRS** functions in a Windows Sockets Vuser script and **LRT** functions in a Tuxedo Vuser script. For details on the protocol-specific Vuser functions, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Standard ANSI C Functions

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, and so forth to your Vuser scripts. You can add standard ANSI C functions to any type of Vuser script. For details, see "Creating Custom Vuser Scripts" in Volume II - the *Protocols user guide*.

Inserting Transactions into a Vuser Script

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.

You can create transactions either during or after recording. To add transactions after recording, use the Transaction editor to graphically mark the steps of a transaction, as described in "Transactions" on page 62. Alternatively, use the **Insert** menu to add **Start Transaction** and **End Transaction** markers.

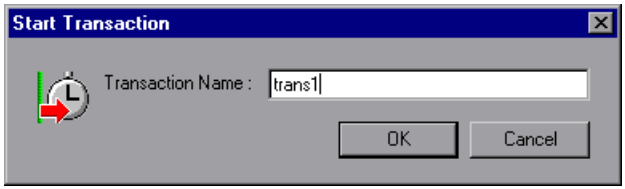
The following sections describe how to create a transaction during recording.

Marking the Beginning of a Transaction

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.

To mark the start of a transaction:

1 While recording a Vuser script, click the **Start Transaction** button on the Recording toolbar. The Start Transaction dialog box opens.



2 Type a transaction name in the Transaction Name box. Transaction names must begin with a letter or number and may contain letters or numbers.

Note: Do not use the following characters: . , : # / \ “ <

Click OK to accept the transaction name. VuGen inserts an lr_start_transaction statement into the Vuser script. For example, the following function indicates the start of the trans1 transaction:

```
lr_start_transaction("trans1");
```

Marking the End of a Transaction

You mark the end of a business process with an end transaction statement.

To mark the end of a transaction:

1 While recording a script, click the **End Transaction** button on the Recording toolbar. The End Transaction dialog box opens.

2 Click the arrow for a list of open transactions. Select the transaction to close.

Click OK to accept the transaction name. VuGen inserts an **lr_end_transaction** statement into the Vuser script. For example, the following function indicates the end of the trans1 transaction:

```
lr_end_transaction("trans1", LR_AUTO);
```

Note: 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.

Inserting Rendezvous Points (LoadRunner only)

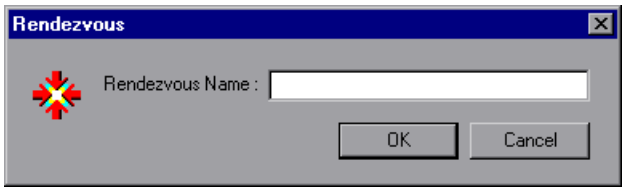
When performing load testing, you need to emulate heavy user load on your system. To accomplish this, you synchronize Vusers to perform a task at exactly the same moment. You configure multiple Vusers to act simultaneously by creating a *rendezvous point*. When a Vuser arrives at the rendezvous point, 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.

Note: Rendezvous points are only effective in *Action* section(s)—not *init* or *end*.

To insert a rendezvous point:

1 While recording a Vuser script, click the **Rendezvous** button on the Recording toolbar. The Rendezvous dialog box opens.



2 Type a name for the rendezvous point in the **Rendezvous Name** box.

Note: The name for the rendezvous point is not case sensitive. For example, the Vuser recognizes Rendezvous1 and rendezvous1 as the same point.

Click OK. VuGen inserts lr_rendezvous into the Vuser script. For example, the following function defines a rendezvous point named rendezvous1:

```
lr_rendezvous("rendezvous1");
```

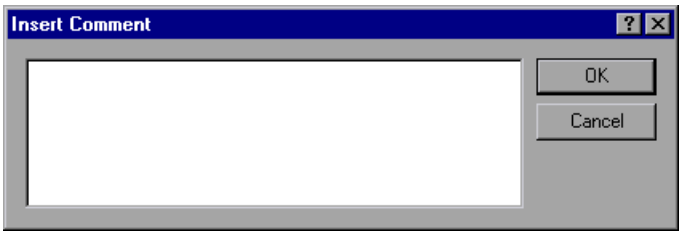
3 To insert rendezvous points into your script after the recording session, select **Insert** > **Rendezvous** from the VuGen toolbar.

Inserting Comments into a Vuser Script

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."

To insert a comment:

1 While recording a script, click the **Comment** button on the Recording tool bar. The Insert Comment dialog box opens.



2 Type the comment into the text box.

3 Click OK to insert the comment and close the dialog box. The text is placed at the current point in the script, enclosed by comment markers. The following script segment shows how a comment appears in a Vuser script:

```
/*
* This is the first query
*/
```

Note: You can insert comments into your script after you complete a recording session, by selecting **Insert** > **Comment** from the VuGen menu.

Obtaining Vuser Information

You can add the following functions to your Vuser scripts to retrieve Vuser information:

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 Availability Center.
lr_whoami	Returns the name of a Vuser executing the script. Not applicable when working with the HP Business Availability Center.

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, refer to the *Online 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. 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 Availability Center, 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:

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 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 directory. This function is useful in preventing output messages from interfering with TCP/IP traffic.
lr_output_message	Sends a message to the Output window 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 Availability Center.
lr_message	Sends a message to the Vuser log and Output window or the Business Process Monitor log files.

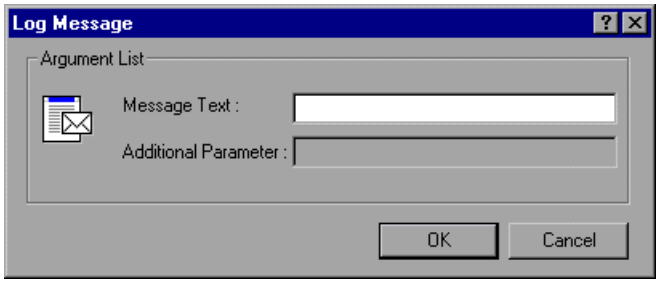
Note: The behavior of the **lr_message**, **lr_output_message**, and **lr_log_message** functions are not affected by the script's debugging level in the Log run-time settings—they will always send messages.

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 an lr_log_message function:

1 Select **Insert** > **Log Message**. The Log Message dialog box opens.



2 Type the message into the **Message Text** box.

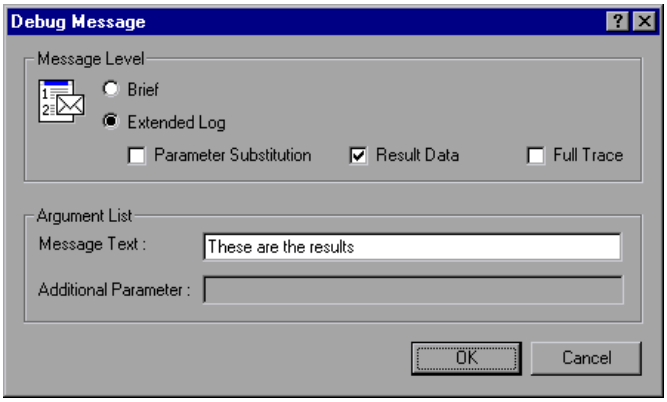
3 Click **OK** to insert the message and close the dialog box. An **lr_log_message** function is inserted at the current point in the script.

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 function:

1. Select **Insert** > **New Step**. The Add Step dialog box opens.
2. Select the **Debug Message** step and click **OK**. The Debug Message dialog box opens.



3. Select a message level, **Brief** or **Extended Log**. If you choose Extended Log, indicate the type of information to log: **Parameter Substitution**, **Result Data**, or **Full Trace**.
4. Type the message into the **Message Text** box.
5. Click **OK** to insert the message and close the dialog box. An **lr_debug_message** function is inserted at the current point in the script.

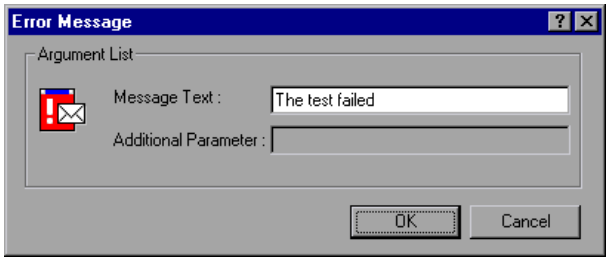
Error and Output Messages

For protocols with a Tree view representation of the script, 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 function:

1 Select **Insert** > **New Step**. The Add Step dialog box opens.

2 Select the **Error Message** or **Output Message** step and click **OK**. The Error Message or Output Message dialog box opens.



3 Type the message into the **Message Text** box.

4 Click **OK** to insert the message and close the dialog box. An **lr_error_message** or **lr_output_message** function is inserted at the current point in the script.

For more information about the message functions, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Handling Errors in Vuser Scripts During Execution

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. For details, see "Error Handling" on page 235.
- 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);
```

Synchronizing Vuser Scripts

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:

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 on using synchronization functions in RTE Vuser scripts, see “Synchronizing RTE Vuser Scripts” in Volume II - the *Protocols user guide*.

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. 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 manually add a think time statement:

1. Place the cursor at the desired location.
2. Choose **Insert** > **Add Step**. The Add Step dialog box opens.
3. Select **Think Time** and click **OK**. The Think Time dialog box opens.

4. Specify the desired think time in seconds and click **OK**.

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

You can use the think time settings to influence how the **lr_think_time** statements operate when you execute a Vuser script. To access the think time settings, select **Vuser** > **Run-time Settings** from the VuGen main menu, and then click the **Think Time** tab. For more information, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Handling Command Line Arguments

You can pass values to a Vuser script at run-time by specifying command line arguments when you run the script. You specify the command line arguments within the Run-Time settings dialog box. For more information, see "Configuring Additional Attributes Run-Time Settings" on page 233.

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 one of the following two formats where the arguments and their values are listed in pairs, after the script name:

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

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

The following example shows the command line string used to repeat script1 five times on the load generator pc4:

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

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

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. 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"));
```

You can restore the string at any time, to determine its original value.

To encrypt a string:

1. For protocols that have tree views, view the script in script view. Choose **View** > **Script View**.
2. Select the text you want to encrypt.
3. Select **Encrypt string (***string***)** from the right-click menu.

To restore an encrypted string:

1. For protocols that have tree views, view the script in script view. Choose **View** > **Script View**.
2. Select the string you want to restore.
3. Select **Restore encrypted string (***string***)** from the right-click menu.

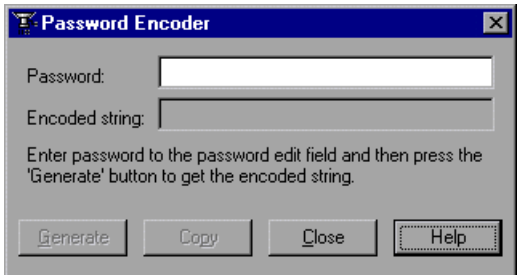
For more information on the **lr_decrypt** function, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Encoding Passwords Manually

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 ensure 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:

1. From the Windows menu, select **Start** > **Programs** > **LoadRunner** > **Tools** > **Password Encoder**. The Password Encoder dialog box opens.



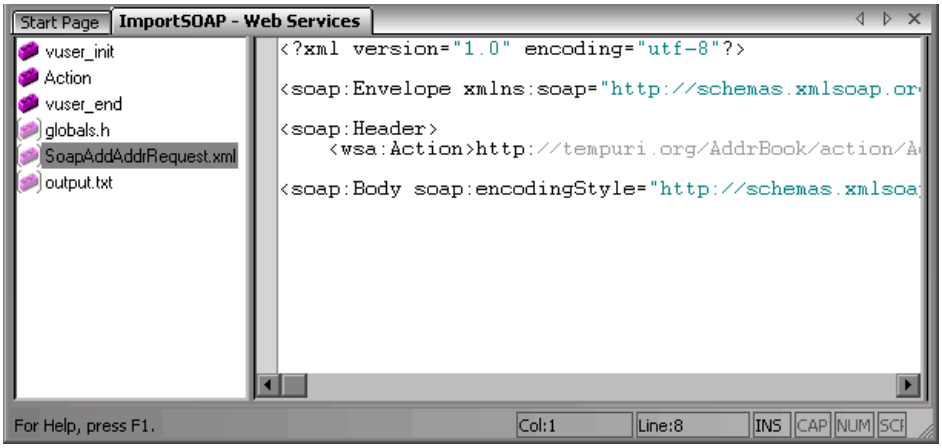
2. Enter the password in the **Password** box.
3. Click **Generate**. The Password Encoder encrypts the password and displays it in the **Encoded String** box.
4. Use the **Copy** button to copy and paste the encoded value into the Data Table.
5. Repeat the process for each password you want to encode.
6. Click **Close** to close the Password Encoder.

Adding Files to the Script

You can add files to your script directory 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:

1. Open the script.
2. Choose **File** > **Add Files to Script**, or right-click the Task pane and select **Add Files to Script** from the right-click menu.
3. Locate the files and click **Open**. VuGen adds the selected files to the script directory and the VuGen editor.
4. Select the file in the left pane to display its contents.



9

Working with VuGen Parameters

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 following information applies to all Vuser Scripts.

About VuGen Parameters

When you record a business process, VuGen generates a Vuser script composed of functions. The values of the arguments in the functions are the actual values used during the recording session.

For example, assume that 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);
```

The resulting Vusers then substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables. For more information about data sources, see "Understanding Parameter Types" on page 149.

Parameterizing a Vuser script has two advantages:

- It reduces the size of the script.
- It provides the ability to test your script with different values. For example, if you want to search a library's database for several titles, you only need to write the submit function once. Instead of instructing your Vuser to search for a specific item, use a parameter. During replay, VuGen substitutes different values for the parameter.

Parameterization involves the following two tasks:

- Replacing the constant values in the Vuser script with parameters
- Setting the properties and data source for the parameters

Understanding Parameter 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, refer to the *Online Function Reference* (**Help** > **Function Reference**).

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 *Online Function Reference* 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, refer to the *Online Function Reference*.

Creating Parameters

You create a parameter by giving it a name, and specifying its type and properties. There is no limit to the number of parameters you can create in a Vuser script.

Step 1: Select the argument that you want to parameterize.

If you are in Script view:

Select the argument that you want to parameterize, and select **Replace with a Parameter** from the right-click menu.

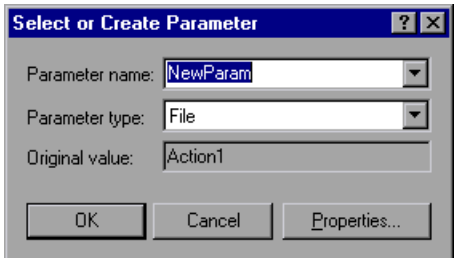
Notes:

- 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 CORBA or General-Java Vuser scripts, you must parameterize complete strings, not parts of a string.

If you are in Tree view:

1. Right-click the step you want to parameterize, and select **Properties** from the menu. The appropriate Step Properties dialog box opens.
2. Click the **ABC** icon next to the argument that you want to parameterize.

   The Select or Create Parameter dialog box opens.



Step 2: Name the parameter.

Type a name for the parameter in the **Parameter name** box. The parameter name is displayed in the script in place of the original argument.

The parameter name should be suitable to the type of information that will replace the parameter during a script run.

For example, if you typically enter a username, then name the parameter Username.

Note: Do not name a parameter *unique*, since this name is used by VuGen.

Step 3: Select a parameter type.

When you create a parameter, you specify the source of the parameter data. This determines the *parameter type*.

Data can be generated internally - such as the date and time, or can be returned as a result of a user-defined function.

Another, very common method for using parameters, is instructing Vusers to take values from an data table or an external file which contains values that the user has defined. These parameters are called File and Table type parameters.

From the **Parameter type** list, select **File**.

For more detailed information about the different parameter types, see "Understanding Parameter Types" on page 149.

Step 4: Define properties for the parameter type.

1. Click **Properties**. The Parameter Properties dialog box opens.
2. Click **Create Table**. A message box opens. Click **OK**.

   VuGen creates a table with one cell containing the argument's original value.
3. To add another value to the table, click **Add Row**, and enter the value.

   Repeat this step to add more values to the table.
4. Click **Close** to close the Parameter Properties dialog box.

For more information, see "Defining Parameter Properties" on page 152.

Step 5: Replace the argument with the parameter.

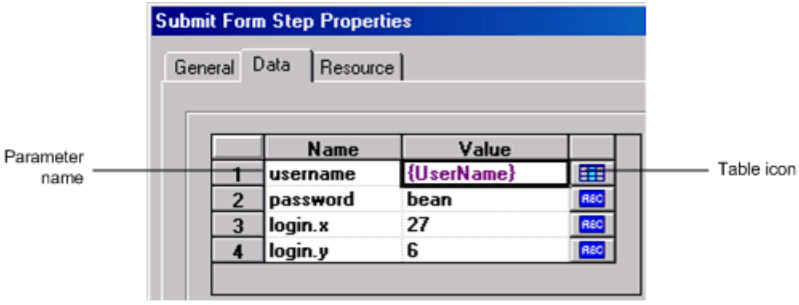
Click **OK** to close the Select or Create Parameter dialog box.

VuGen replaces the selected string in your script with the name of the parameter, surrounded by curly or round brackets.

Note: The default parameter braces are either curly or angle brackets, depending on the protocol type. You can change the parameter braces from the Parameterization tab in the General Options dialog box (select **Tools** > **General Options**). For more information, see "Setting Parameterization Options" on page 159.

In Tree view, VuGen replaces the **ABC** icon with the table icon.

In the following example, the original **username** value was **jojo**. It has been replaced with the parameter **{UserName}**.



Understanding Parameter Types

When you create a parameter, you specify the source for the parameter data. You can specify any one of the following data source types:

- File or Table Parameter Types
- XML Parameter Types
- Internal Data Parameter Types
- User-Defined Function Parameters
- BPT Type Parameters

File or Table Parameter Types

Data that is contained in a file—either an existing file or one that you create with VuGen or MS Query. A very common method for using parameters, is instructing Vusers to take values from an external file or a data table.

Data Files

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.

Data Tables

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 SAPGUI Vusers where the **sapgui_fill_data** function fills the table cells.

For information about defining data file or data table parameter properties, see Chapter 10, “File, Table, BPT, and XML Parameter Types.”

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. It is recommended to 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.

For information about defining Internal Data parameter properties see "Setting Properties for Internal Data Parameter Types" on page 188.

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, it is recommended 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 directory is the default dynamic library path. You can add your library to this directory.

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;}
```

For information about defining User-Defined Function properties, see "Setting Properties for User-Defined Functions" on page 198.

BPT Type Parameters

You typically use BPT (Business Process Test) type parameters to share parameters between business components in Quality Center. In the Parameter Properties dialog box, you can configure properties such as Input/Output, value, data type and description. For information on setting parameter properties in Service Test, see "Defining Parameter Properties" on page 152.

For more information, see the section on Business Process Tests or refer to the *Business Process Testing User's Guide*.

Defining Parameter Properties

You can define a parameter's properties in the Parameter Properties dialog box or in the Parameter List dialog box.

To define parameter properties in the Parameter Properties dialog box:

1 Open the Parameter Properties dialog box.

You open the Parameter Properties dialog box in one of the following ways:

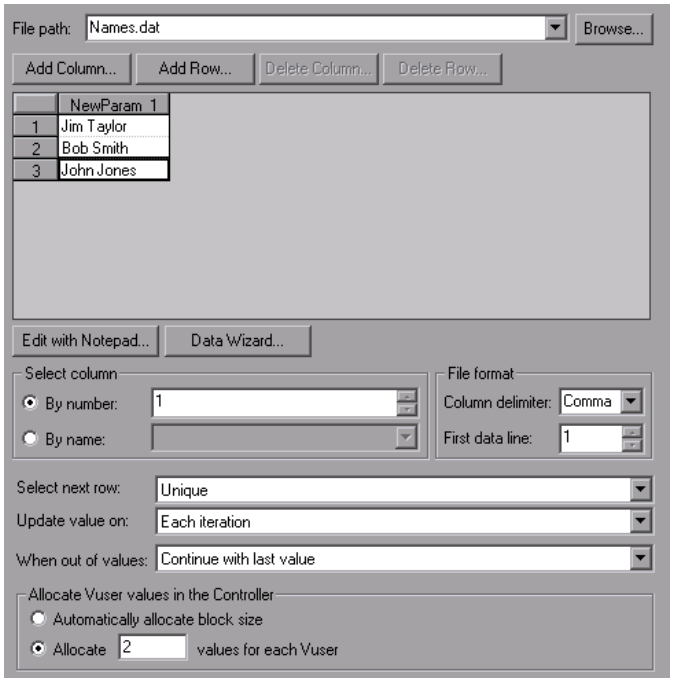
➤ When you create a new Parameter as described in "Creating Parameters" on page 146, you click **Properties** in the Select or Create Parameter dialog box to open the Parameter Properties dialog box.

➤ In Script view, select the parameter, and choose **Parameter Properties** from the right-click menu.

➤ In Tree view, right-click the step containing the parameter whose properties you want to define, and select **Properties**. The Step Properties dialog box for the selected step opens.

Click the table icon beside the parameter whose properties you want to define, and select **Parameter Properties** from the pop-up menu.

In the following example, the properties of a **file** type parameter are displayed:



2 Define the parameter properties.

➤ To define properties for File and Table type parameters, see Chapter 10, “File, Table, BPT, and XML Parameter Types.”

➤ To define properties for internal data parameter types, see “Setting Properties for Internal Data Parameter Types” on page 188.

➤ To define BPT properties, see "Setting Properties for BPT Type Parameters" on page 171.

➤ To define properties for user-defined functions, see "User-Defined Function Parameters" on page 151.

3 Close the Parameter Properties dialog box.

Click **Close** to close the Parameter Properties dialog box.

To define parameter properties in the Parameter List dialog box:

Click the **Parameter List** button, or select **Vuser** > **Parameter List**. Select a parameter to show its properties.

For more information, see "Using the Parameter List" on page 157.

Using Existing Parameters

When you create a parameter, VuGen stores it in a parameter list. You can use an existing parameter to replace an argument, or to replace multiple occurrences of an argument.

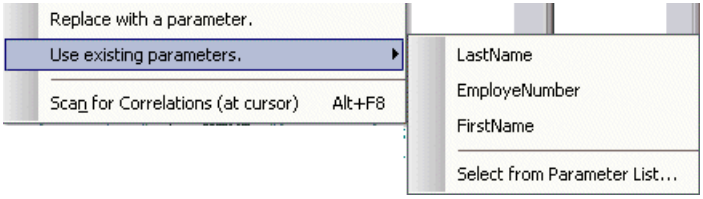
Replacing Strings Using Pre-defined Parameters

You can assign a pre-defined parameter to an argument.

To replace a string with a pre-defined parameter:

1 Enter Script view.

2 Right-click on the argument that you want to parameterize, and select **Use existing parameters**. A submenu opens.



3 Use one of the following options to select a parameter:

➤ Select a parameter from the submenu list.

➤ Choose **Select from Parameter List** to open the Parameter List dialog box, and select a parameter from the left pane.

Using the **Parameter List** is convenient when you want to replace an argument with a previously defined parameter and, at the same time, view or modify that parameter's properties. For details on using the Parameter List, see "Using the Parameter List" on page 157.

Replacing Multiple Occurrences

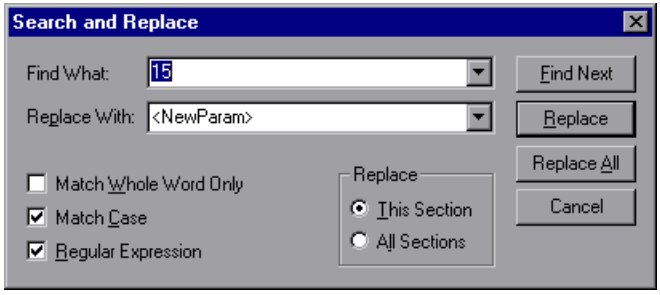
When you create a parameter, the system remembers the original value of the argument. You can use the **Search and Replace** function to replace selected or all occurrences of the same argument with the same parameter or another existing parameter.

To replace multiple occurrences of an argument with a specific parameter:

1 Right-click a parameter and choose **Replace more occurrences** from the menu.

The Search and Replace dialog box opens. The **Find What** box displays the value or argument you want to replace. The **Replace With** box displays the parameter name in brackets.

2 Select the appropriate check boxes for matching whole words or case. To search with regular expressions (., !, ?, +, and so forth.) select the **Regular Expressions** check box.



3 Click **Replace** or **Replace All**.

In the above example, all arguments of value **15** are replaced with the parameter, **{NewParam}**.

Note: Use caution when using **Replace All**, especially when replacing number strings. VuGen changes all occurrences of the string.

Restoring Original Strings

VuGen lets you undo the parameterization and restore the originally recorded argument.

To restore a parameter to its original value:

- In Script view, right-click on the parameter and select **Restore original value**.
- In Tree view:
  - Right-click on the step that contains the parameter and click **Properties**.
  -  Click the table icon next to the parameter that you want to restore to its original value, and select **Undo Parameter**.

The original argument is restored.

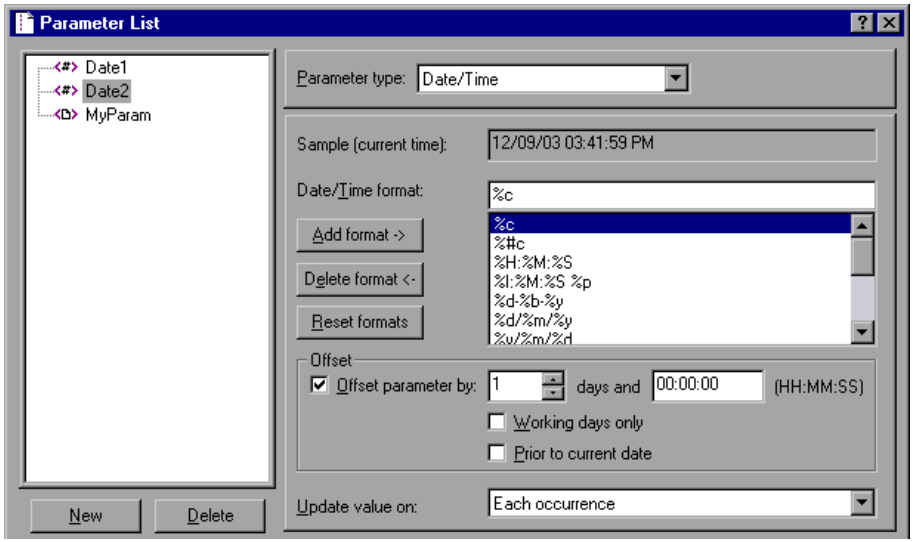
Using the Parameter List

You use the Parameter List to examine all of the parameters, create new parameters, delete parameters, or modify a parameter properties.

To view the Parameter List and view a parameter's properties:

Click the **Parameter List** button, or select **Vuser** > **Parameter List**. Select a parameter to show its properties.

In the following example, the properties of a **Date/Time** type parameter are displayed:



To modify a parameter's properties:

Select the parameter from the parameter tree on the left, and edit the parameter's type and properties in the right pane.

For more information on setting parameter properties, see Chapter 11, "Setting Parameter Properties," and Chapter 10, "File, Table, BPT, and XML Parameter Types."

To create a new parameter:

1. In the Parameter List dialog box, click **New**. The new parameter appears in the parameter tree with a temporary name.
2. Type a name for the new parameter, and press Enter.

Note: Do not name a parameter *unique*, since this name is used by VuGen.

---

3. Set the parameter's type and properties.
4. Click **Close** to close the Parameter List dialog box.

Note: VuGen creates a new parameter, but does not automatically replace any selected string in the script.

To delete an existing parameter:

1. Select the parameter from the parameter tree, and click **Delete**. The Delete Parameter dialog box opens.
2. If you want to delete the parameter file from the disk, select **Delete parameter data file from disk**.
3. Click **Yes**.
4. If you selected **Delete parameter data file from disk**, VuGen sends a warning message. Click **Yes** to confirm your action.

Setting Parameterization Options

You set the parameter options in the Parameterization tab of VuGen's General Options window. These options refer to:

- Parameter Braces
- Global Directory

Parameter Braces

When you insert a parameter into a Vuser script, VuGen places parameter braces on either side of the parameter name. The default braces for a Web or WAP script are curly brackets, for example:

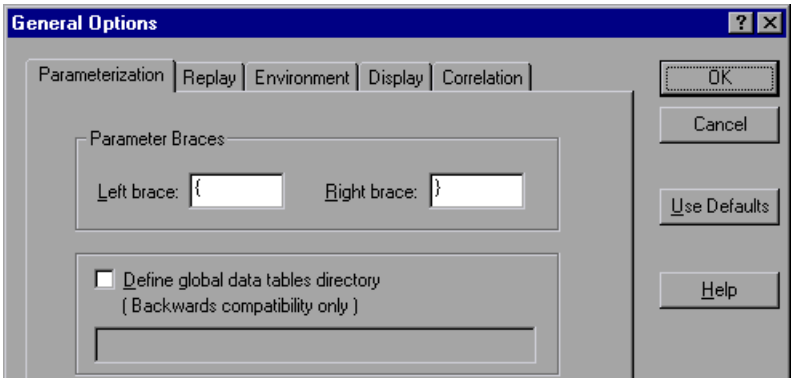
```
web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
        "value={Book_Title}",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
```

You can change the style of parameter braces by specifying a string of one or more characters. All characters are valid with the exception of spaces.

Note: The default parameter braces are angle or curly brackets, depending on the protocol type.

To change the parameter brace style:

1. Select **Tools** > **General Options** in VuGen. The General Options dialog box opens.
2. Select the **Parameterization** tab and enter the desired brace.



3. Click **OK** to accept the settings and close the dialog box.

Global Directory

This option is provided only for backward compatibility with earlier versions of VuGen. In earlier versions, (4.51 and below), when you created a new data table, you specified local or global. A local table is saved in the current Vuser script directory and is only available to Vusers running that script. A global table is available to all Vuser scripts. The global directory can be on a local or network drive. Make sure that the global directory is available to all machines running the script. Using the General Options dialog box, you can change the location of the global tables at any time.

In newer versions of VuGen, you specify the location of the data table either in the Parameter Properties dialog box or in the Parameter List dialog box. VuGen is able to retrieve the data from any location that you specify, be it the default script directory or another directory on the network. For more information, see “Data Files” on page 150.

By default, the **Define global data tables directory** option is disabled.

To set the global directory:

1. Select **Tools** > **General Options**. The General Options dialog box opens.
2. Select the **Parameterization** tab.
3. Select the **Define global data tables directory** check box, and specify the directory containing your global data tables.
4. Click **OK** to accept the settings and close the dialog box.

10

File, Table, BPT, and XML Parameter Types

A very common method for using parameters, is instructing Vusers to take values from an data table or an external file. The data is contained either in an existing file or in a file that you create with VuGen or MS Query.

The following information applies to all Vuser Scripts.

Selecting or Creating Data Files or Data Tables

When you create a File or Table parameter you have to create a .dat file to store the data, or open an existing one. Then you define the other properties for the parameter, such as how the Vuser should assign values to the parameter.

You can create a new data table or select an existing data source from the File Path list.

To select a source file or table for your data:

1 Open the Parameter Properties dialog box or the Parameter List.

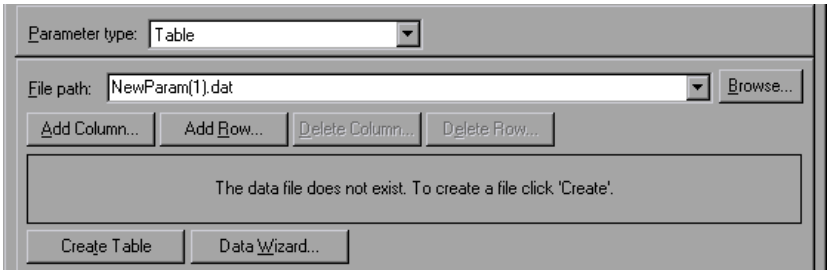
For instructions, see "Defining Parameter Properties" on page 152.

2 Select a table or create a new one.

➤ If there are no tables (**.dat** files) listed in the file path list, or you want to create a new table, click **Create Table**. VuGen creates a new table with one cell, displaying the original value of the argument in the first column of the table.

➤ To open an existing data file, type the name of the **.dat** file in the **File path** box or choose a name from the drop-down list.

Alternatively, click **Browse** to specify the file location of an existing data file. By default, all new data files are named <*parameter_name*>**.dat** and are stored in the script's directory.



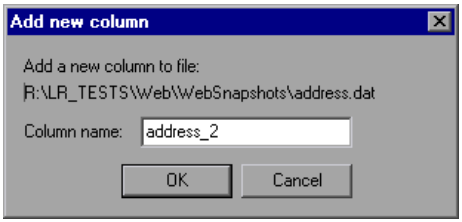
VuGen opens the data file and displays the first 100 rows. To view all of the data, click **Edit with Notepad** and view the data in a text editor.

Note: You can also specify a global directory. Global directories are provided only for backward compatibility with earlier versions of VuGen. For more information, see "Global Directory" on page 160.

➤ To import data from an existing database, click **Data Wizard** and follow the wizard's instructions. For more information, see "Importing Data from an Existing Databases" on page 166.

3 Add columns and rows to the table.

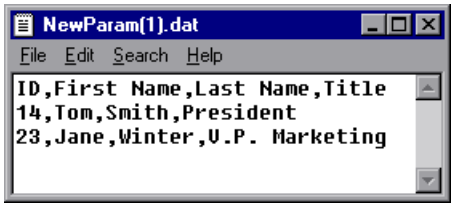
➤ To add additional columns to the table, choose **Add Column**. The Add new column dialog box opens. Enter a column name and click **OK.**



➤ To add additional rows to the table, choose **Add Row**.

4 Edit the data file.

- Click within any cell to enter a value.
- To edit the data file from within Notepad, click **Edit with Notepad**. 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).



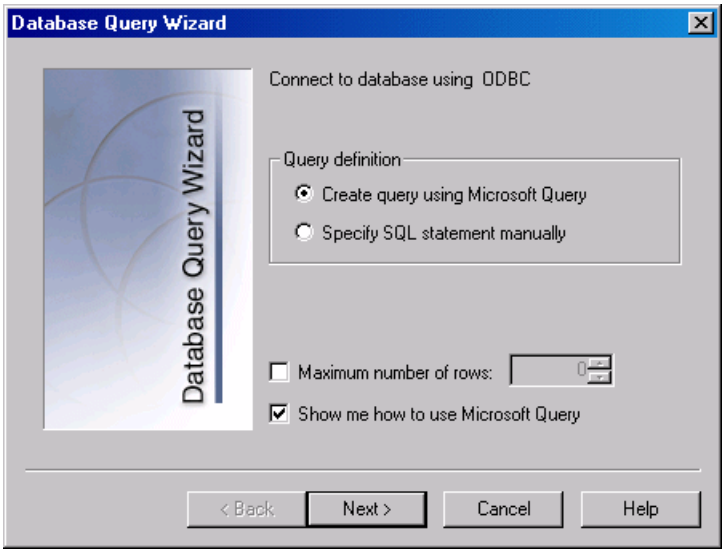
Importing Data from an Existing Databases

VuGen allows you to import data from a database for use with parameterization. You can import the data in one of two ways:

- Creating a New Query
- Specifying an SQL Statement

VuGen provides a wizard that guides you through the procedure of importing data from a database. In the wizard, you specify how to import the data—create a new query via an MS Query or by specifying an SQL statement. After you import the data, it is saved as a file with a *.dat* extension and stored as a regular parameter file.

To begin the procedure of importing a database, click **Data Wizard** in the Parameter List dialog box (**Vuser** > **Parameter List**). The Database Query Wizard opens.



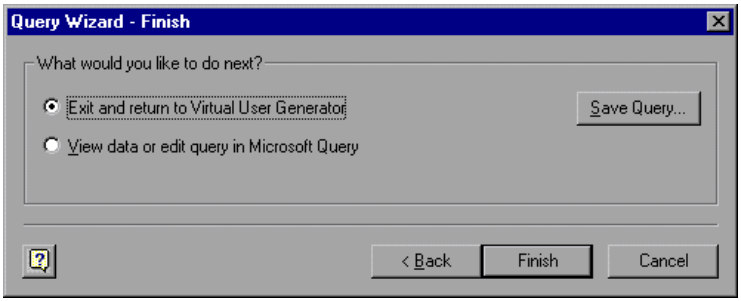
Creating a New Query

You use Microsoft's Database Query Wizard to create a new query. This requires the installation of MS Query on your system.

To create a new query:

1. 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, choose **Exit and return to the Virtual User Generator** and click **Finish**. The database records appear in the Parameter Properties box as a data file.



To edit and view the data in MS Query, choose **View data or edit in Microsoft Query**.

5 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 170.

Specifying an SQL Statement

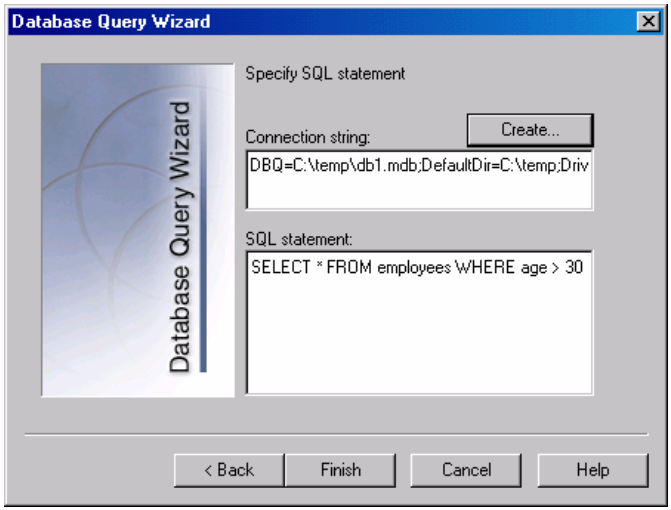
To specify a database connection and SQL statement:

1 Select **Specify SQL Statement**. 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 **SQL** box, type or paste 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.

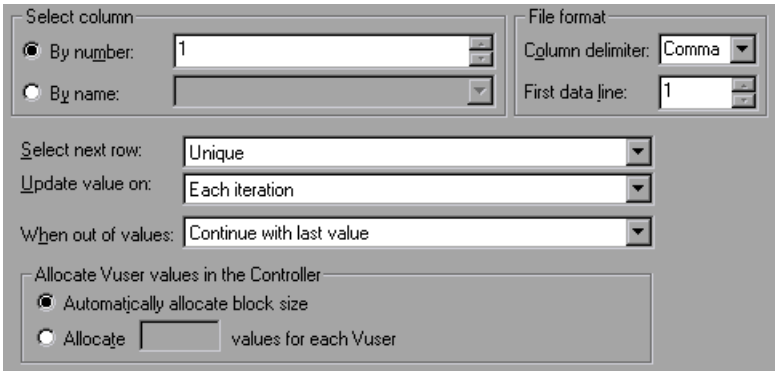
6 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 170.

After creating table or file data, you set the assignment properties. The properties specify the columns and rows to use, and whether to use the data randomly or sequentially. You set the properties separately for the File and Table type parameters.

Note: You can also set the properties for a parameter from the Parameter List dialog box. In the left pane, select the parameter and then specify its properties in the right pane. See "Using the Parameter List" on page 157.

Setting Properties for File Type Parameters

After you select a source of data, you set the assignment properties for your file. These properties instruct VuGen how to use the data. For example, they indicate which columns to use, how often to use new values, and what do to when there are no more unique values.



To set the File parameter properties:

1. Specify the column in the table that contains the values for your parameter. In the **Select column** section, specify a column number or name.

   To specify a column number, select **By number** and the column number. 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.

   To specify a column name, select **By name** and choose the column name from the list. The column header is the first row of each column (row 0). If column numbers might change, or if there is no header, use the column name to select a column.

2. In the **Column delimiter** box of the **File format** section, enter the column delimiter—the character used to separate the columns in the table. You can specify a comma, tab, or space.

3. In the **First data line** box of the **File format** section, select 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.

4. Select a Data Assignment method from the **Select next row** list 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 "Choosing Data Assignment Methods for File/Table Parameters" on page 175.
5. Choose an update option from the **Update value on** list. 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 177.
6. If you chose **Unique** as the Data Assignment method (in step 4):
   - **When out of values.** Specify what to do when there is no more unique data: **Abort the Vuser**, **Continue in a cyclic manne***r*, 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. Choose **Automatically allocate block size** or **Allocate** x **values for each Vuser.** For the second option, specify the number of values to allocate.

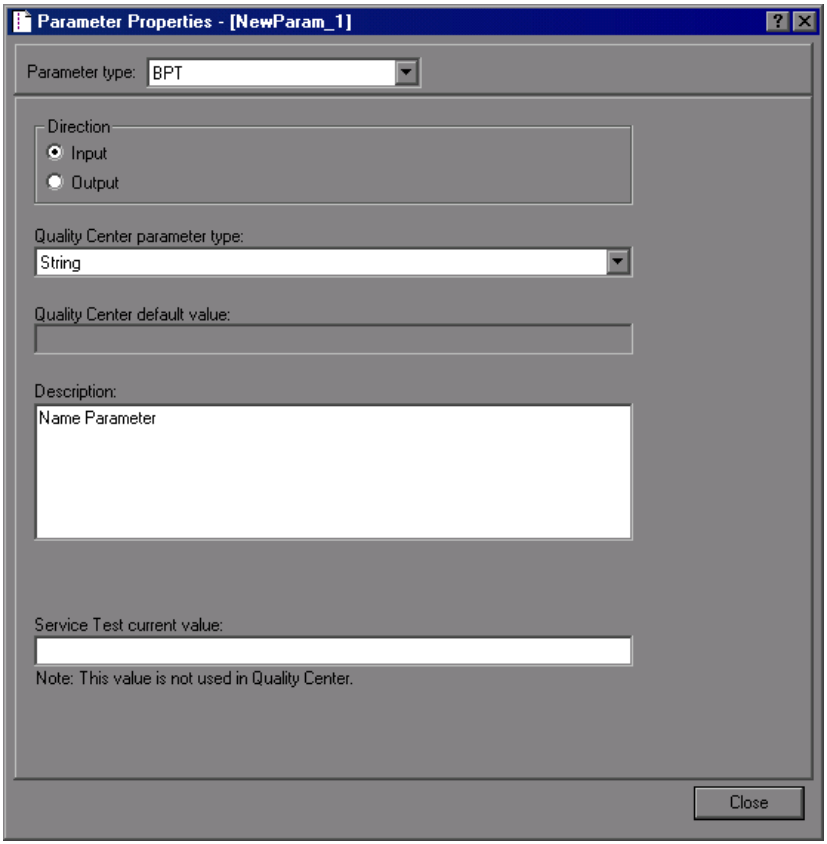
     To track this occurrence, enable the **Extended Log** > **Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <*table_name*>".

Setting Properties for BPT Type Parameters

You set BPT type parameters to allow you to share values between business components in Quality Center. You can set the following values:

- **Direction.** Input or Output. Select **Input** to use the parameter as an input parameter for the current component. Choose **Output** to assign the current component's parameter as an output value.
- **Quality Center parameter type.** Select a data type for which the parameter will be used in Quality Center: String, Boolean, Date, Number, or Password.
- **Description.** A textual description of the parameter.

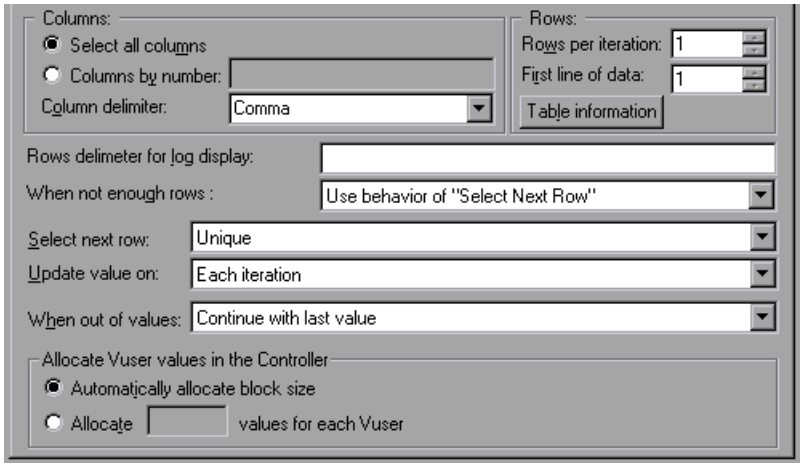
- **Quality Center default value.** (Input only) The default value of the parameter in Quality Center, if it had already been created in Quality Center.
- **Service Test current value.** The current value of the parameter in Service Test. This value is not used by Quality Center.



Specify the desired values and click **Close**. When you save your script, the parameter is saved in Quality Center, provided that you have an open connection to the server.

Setting Properties for Table Type Parameters

After you select a table of data, you set its assignment properties. These properties instruct VuGen how to use the table data. For example, they indicate which columns and rows to use, how often to use them, and what to do when there are no more unique values.



To set the Table parameter properties:

1. Specify the columns in the table that contains the values for your parameter. In the **Columns** section, specify which columns you want to use.

   To choose all columns, 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.

2. 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.

3 In the **Rows** section, specify how many rows to use per iteration in the **Rows per iteration** box.

Note: 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.

---

4 In the **First line of data** box, select the first line of data to be used during script execution. To begin with the first line after the header, enter 1. To display information about the table, including how many rows of data are available, click **Table information**.

5 Specify a row delimiter for your data presentation in the **Rows delimiter for log display** box. 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.

6 In the **When not enough rows** box, specify a handling method when there are not enough rows in the table for the iteration. For example, assume that the table you want to fill has 3 rows, but your data only has two rows. Choose **Parameter will get less rows than required** to fill in only two rows. Choose **Use behavior of "Select Next Row"** to loop around and get the next row according the method specified in the **Select next row** box—**Random** or **Sequential**.

7 Select a Data Assignment method from the **Select next row** list to instruct the Vuser how to select the table data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Parameters" on page 175.

8 Choose an Update method from the **Update value on** list. The options are **Each Iteration** or **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 177.

9 If you chose to assign data using the **Unique** method:

- **When out of values.** Specify how to proceed 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** (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. Choose **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*>".

Choosing Data Assignment Methods for File/Table 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 are available:

- Sequential
- Random
- Unique

Sequential

The **Sequential** method 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

The **Random** method assigns a random value from the data table to each Vuser at the start of the test run.

When running a scenario or Business Process Monitor profile, 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 refer to the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Unique

The **Unique** method assigns a unique sequential value to the parameter for each Vuser.

In this case you must make sure there is enough data in the table for all the 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 there are not enough values in the data table, you can instruct VuGen how to proceed. For more details, see "Setting Properties for File Type Parameters" on page 170, or "Setting Properties for Table Type Parameters" on page 173.

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 *HP LoadRunner Controller User's Guide*.

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 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

- If you chose to assign data using the **Sequential** method, then:
  - If you choose to 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 choose to 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 choose to update **Once**, all Vusers take Kim for all iterations.

  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.
- If you chose to assign data using the **Random** method, then:
  - If you choose to update on **Each iteration**, the Vusers use random values from the table for each iteration.
  - If you choose to update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
  - If you choose to update **Once**, all Vusers take the first randomly assigned value for all the iterations.
- If you chose to assign data using the **Unique** method, then:
  - If you choose to 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 choose to update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.
  - If you choose to 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.

Setting Properties for 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 type parameter and assign a different value set for each iteration. The XML parameter type also supports complex schema types such as arrays and <any> elements.

Note: The XML parameter type does not support <choice> schema type elements.

This section describes:

- Creating New XML Parameters
- Modifying XML Parameter Properties

Creating New XML Parameters

You can create XML Parameters in the Parameter List or from an existing recorded script.

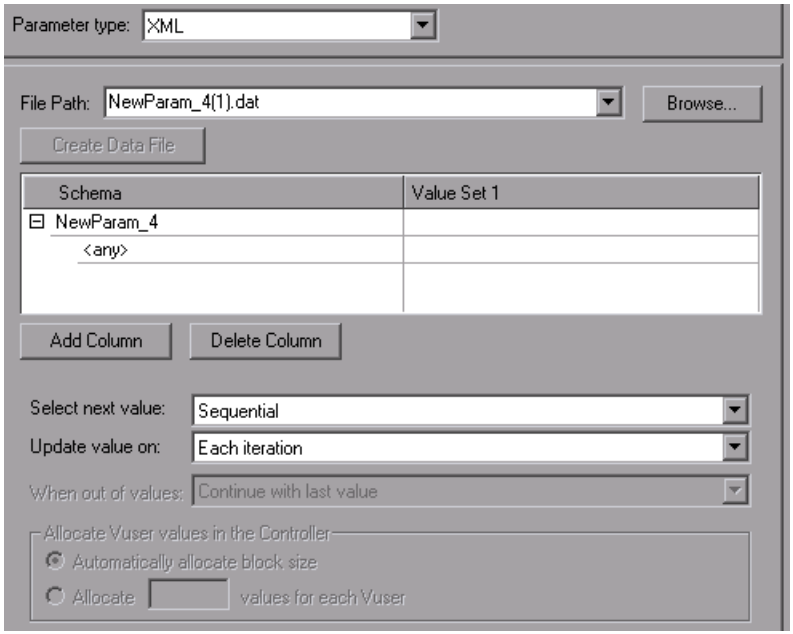
Creating New XML Parameters

To create a new XML parameter:

1 **Open the Parameter List and create a new parameter of type XML.**

For instructions, see "Defining Parameter Properties" on page 152.

2 Define a schema for the XML parameter.



a In the Schema column, right-click **<any>** and select one of the following options:

- **Add New Sub-Element.** Inserts a sub-element under the selected element.
- **Insert Array Element.** Inserts a new array element with the same array structure as the array from which you selected this option.
- **Delete Array Element.** Deletes a complete array element.
- **Copy Row XPath**. Copies the full XML path of the selected element to the clipboard.

3 Define value sets for the XML parameter.

In the **Value Set** column, insert values corresponding to the schema you created.

To insert more value sets, click **Add Column**, and insert another set of values in the new column.

4 Define a data assignment method and an update option for the parameter.

a 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 "Choosing Data Assignment Methods for File/Table Parameters" on page 175.

b 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 177.

c 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 manne***r*, 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. Choose **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>**.

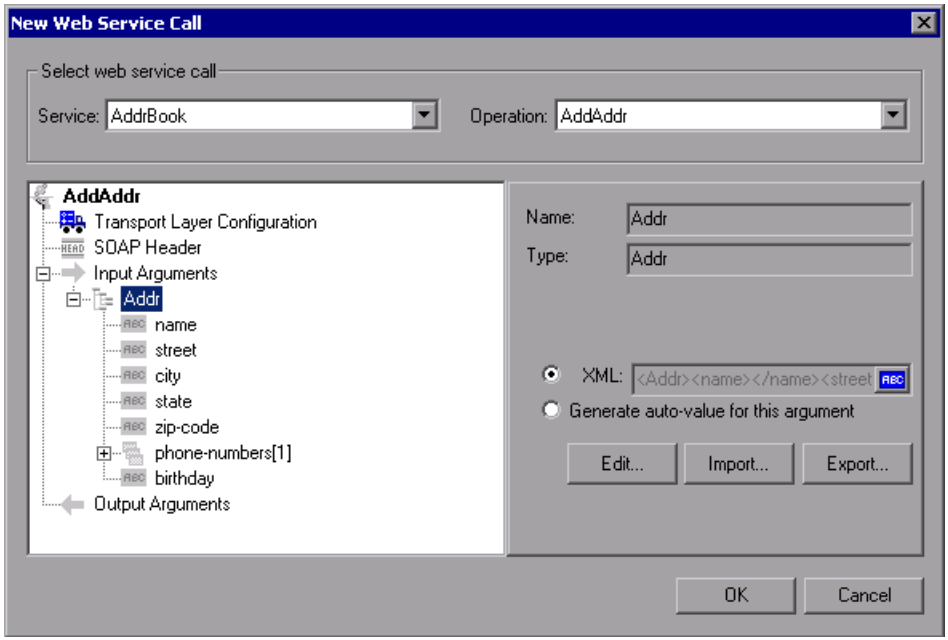
5 Add the parameter to the Parameter List.

In the Parameter Properties dialog box, click **Close**.

Creating XML Parameters From a Web Service Script

When you add a Web Service call, the New Web Service Call dialog box shows the array elements of the **Input Arguments**. You can define a single XML parameter that will contain values for all of the array elements.

Note: When saving an array to a parameter, the number of array elements per parameter is constant. If you want to run multiple iterations, with each iteration using a different number of array elements, you need to define separate parameters, each containing the desired number of array elements.



To parameterize an array:

1 Select the root element of the complex data structure.

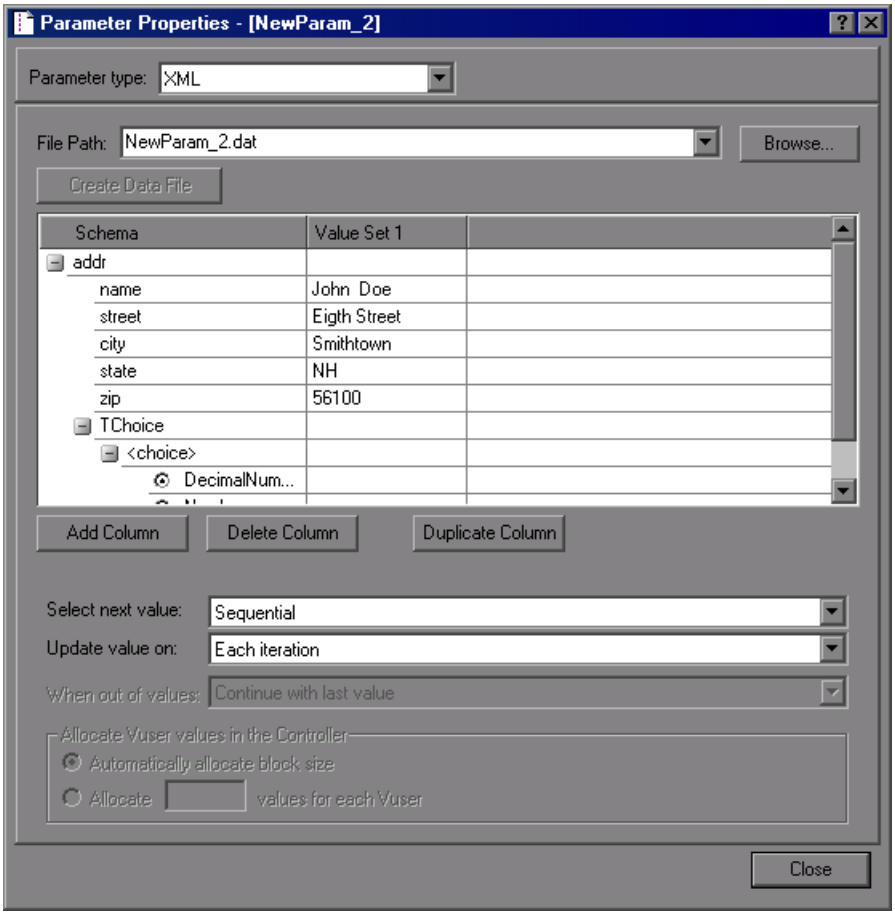
The right pane displays the argument's details.

2 Select **XML** and click the **ABC** button. 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**. The Parameter Properties dialog box opens.



- ➤ The File Path box displays the name of the .dat file that stores the argument data sets. The .dat file is assigned the same name that you gave to the parameter.
- ➤ In the table, the **Schema** column displays the operation arguments that were listed in the tree when you added the Web Service call.

6 In the second column, **Value Set 1**, enter a set of values, each value corresponding to the set of arguments in the **Schema** column.

To add another value set, click **Add Column**, and enter the values in the new column. To duplicate an existing column, place the cursor within an existing column and click **Duplicate Column**.

7 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 "Choosing Data Assignment Methods for File/Table Parameters" on page 175.

8 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 177.

9 If you chose **Unique** as the data assignment method (in step 7 above) 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 manne***r*, 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. Choose **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>**.

10 Add the parameter to the Parameter List.

a In the Parameter Properties dialog box, click **Close**.

b In the Select or Create Parameter dialog box, click OK.

The list of input arguments is replaced by the parameter name, and ABC button is replace by a table icon button which you can click to edit the parameter properties or un-parameterize the parameter.

Modifying XML Parameter Properties

After you have created an XML parameter, you can modify the value sets from the Web Service's Step Properties tab.

To modify XML parameter properties:

1. In the Web Service script's tree view, click the **Step Properties** tab.
2. Under **Input Arguments**, select the XML parameter. The right pane displays the parameter details.
3. To modify the XML parameter properties, click the table icon button adjacent to the **XML** box and select **Parameter Properties**.
4. Modify the parameter properties as described from step 5 in "Creating New XML Parameters" on page 180.

11

Setting Parameter Properties

A parameter is defined according to the type of information it replaces.

The following information applies to all Vuser Scripts.

About Setting Parameter Properties

When you define a parameter's properties, you specify the source for the parameter data. You define properties for any one of the following data source types:

Internal Data Parameter Types	Data that is generated internally by the Vuser: Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.
User-Defined Functions	Data that is generated using a function from an external DLL.
Data Files and Data Tables	Data that is contained in a file—either an existing file or one that you create with VuGen or MS Query.

This chapter describes how to assign properties to Internal Data and User-Defined Function parameters.

For information on defining properties for Table or File type parameters, see Chapter 10, "File, Table, BPT, and XML Parameter Types."

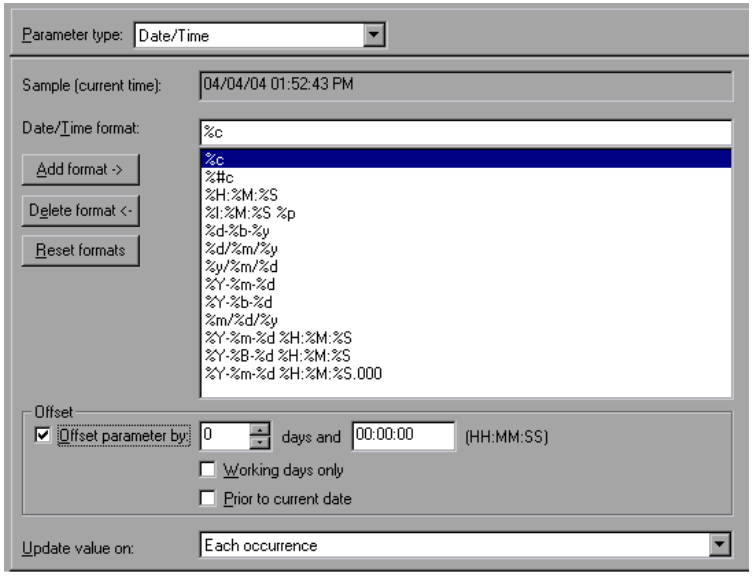
Setting Properties for Internal Data Parameter Types

This section discusses setting the properties for data that is generated internally by the Vuser. Internal data includes data such as:

- Date/Time
- Group Name
- Iteration Number
- Load Generator Name
- Random Number
- Unique Number
- Vuser ID

Date/Time

Date/Time replaces the parameter with the current date and/or time. To specify a date/time format, you can select a format from the format list or specify your own format. The format should correspond to the date/time format recorded in your script.



VuGen lets you 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. If you want to test your application for a future time, you specify a time offset. You can specify a forward, future offset (default) or a backward offset, a date or time that already passed. In addition, you can instruct VuGen to use date values for work days only, excluding Saturdays and Sundays.

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

Symbol	Description
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)

To set the properties for Date/Time parameters:

1. Select one of the existing date/time formats or create a new format. You can view a sample of how VuGen will display the value, in the **Sample (Current time)** box. For information on customizing parameter formats, see "Customizing Parameter Formats" on page 199.

2. To set a date and time offsets, select **Offset Parameter by** and specify the desired offset for the date and time values.

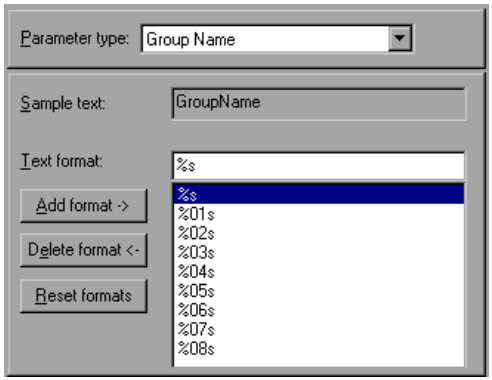
   To instruct VuGen to use working day dates only, excluding weekends, select **Working days only**. To indicate a negative offset to test a date prior to the current, select **Prior to current date**.

3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 200.

4. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Group Name

Group Name replaces the parameter with the name of the Vuser Group. You specify the name of the Vuser Group when you create a scenario. When you run a script from VuGen, the Group name is always *None*.

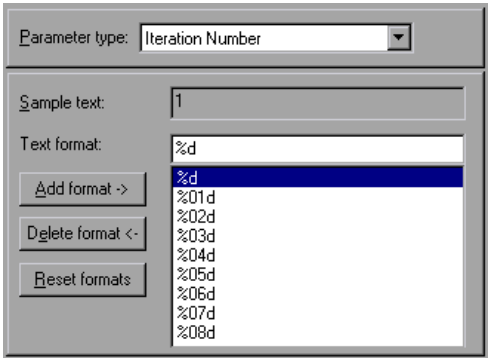


To set properties for the Group Name parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 199.
2. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Iteration Number

Iteration Number replaces the parameter with the current iteration number.

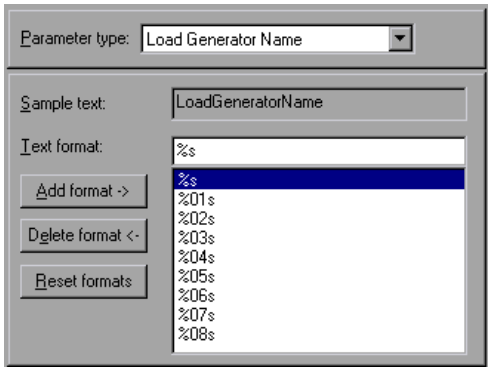


To set the properties for the Iteration Number parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 199.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

Load Generator Name

Load Generator Name replaces the parameter with the name of the Vuser script's load generator. The load generator is the computer on which the Vuser is running.



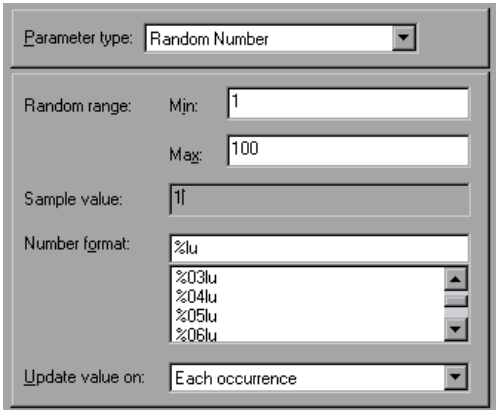
To set the properties for the Load Generator Name parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 199.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

Random Number

Random Number replaces the parameter with a random number. You set a range of numbers by specifying minimum and maximum values.

You can use the Random Number parameter type to sample your system's behavior within a possible range of values. For example, to run a query for 50 employees, where employee ID numbers range from 1 through 1000, create 50 Vusers and set the minimum to 1 and maximum to 1000. Each Vuser receives a random number, from within the range of 1 to 1000.



To set the properties for the Random Number parameter type:

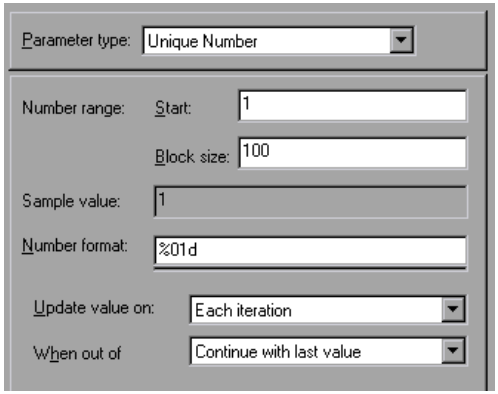
1. Enter a range defining the set of possible parameter values. You specify minimum and maximum values for the range of random numbers.
2. Select a **Number format**, indicating the length of the random number. Specify **%01lu** (or **%lu**) for one digit, **%02lu** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 200.
4. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Unique Number

Unique Number replaces the parameter with a unique number.

When you create a Unique Number type parameter, you specify a start number and a block size. The block size indicates the size of the block of numbers assigned to each Vuser. Each Vuser begins at the bottom of its range and increments the parameter value for each iteration. For example, if you set the Start number at 1 with a block of 500, the first Vuser uses the value 1 and the next Vuser uses the value 501, in their first iterations.

The number of digits in the unique number string together with the block size determine the number of iterations and Vusers. For example, if you are limited to five digits using a block size of 500, only 100,000 numbers (0-99,999) are available. It is therefore possible to run only 200 Vusers, with each Vuser running 500 iterations.



You can also indicate what action to take when there are no more unique numbers in the block: **Abort Vuser**, **Continue in a cyclic manner**, or **Continue with last value** (default).

You can use the Unique Number parameter type to check your system's behavior for all possible values of the parameter. For example, to perform a query for all employees, whose ID numbers range from 100 through 199, create 100 Vusers and set the start number to 100 and block size to 100. Each Vuser receives a unique number, beginning with 100 and ending with 199.

Note: VuGen creates only one instance of Unique Number type parameters. If you define multiple parameters and assign them the Unique Number Parameter type, the values will not overlap. For example, if you define two parameters with blocks of 100 for 5 iterations, the Vusers in the first group will use 1, 101, 201, 301, and 401. The Vusers in the next group, using the second parameter will use 501, 601, 701, 801, and 901.

After you define a Unique parameter, you can use it in other scripts by pointing to the same parameter file. The parameter retains all of the properties that you assigned to the original parameter.

To set the properties for the Unique Number parameter type:

1. Enter a start number and the desired block size. For example, if you want 500 numbers beginning with 1, specify 1 in the **Start** box, and 500 in the **Block size per Vuser** box.
2. Select a Number format, indicating the length of the unique number. Specify **%01d** (or **%d**) for one digit, **%02d** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 200.
4. Indicate what to do when there are no more unique values, in the **When out of values** box: **Abort Vuser**, **Continue in cyclic manner**, or **Continue with last value**.

Note: (For LoadRunner users!)

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.

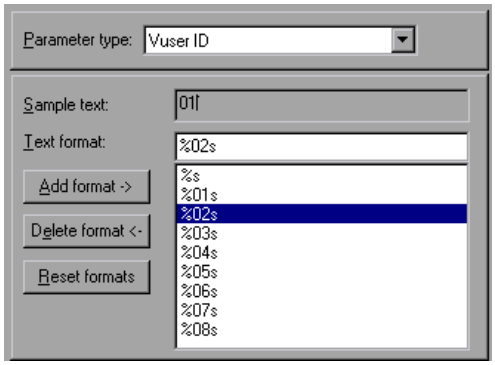
---

5 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Vuser ID

Note: This parameter type applies primarily to LoadRunner users.

Vuser ID replaces the parameter with 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.

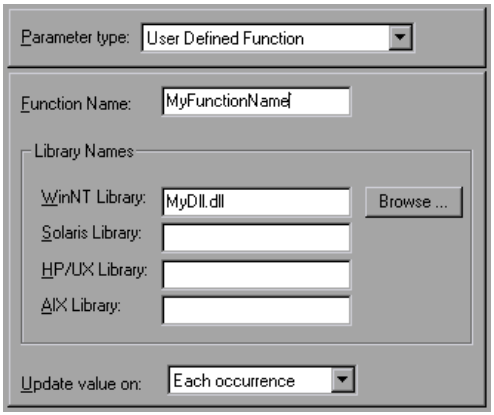
To set the properties for the Vuser ID parameter type:

1 Select one of the available formats or create a new one. You select a format to specify the length and structure of the parameter string. For details, see "Customizing Parameter Formats" on page 199.

2 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Setting Properties for User-Defined Functions

In the Parameter Properties dialog box, select **User Defined Function** from the **Parameter type** list.



To set the properties for user-defined functions:

1. Specify the function name in the **Function Name** box. Use the name of the function as it appears in the DLL file.
2. In the **Library Names** section, specify a library in the relevant **Library** box. If necessary, locate the file using the **Browse** command.
3. Select an update method for the values. For more information on update methods for user-defined functions, see "Selecting an Update Method" on page 200.

Customizing Parameter Formats

For most data types, you can customize a format for a parameter by selecting an existing format or specifying a new one.

Note: The parameter format should match the recorded values. If the format of the parameter differs from the format of the original recorded value, the script may not run correctly.

The format specifies the length and structure of the resulting parameter string. The resulting parameter string is the actual parameter value together with any text that accompanies the parameter. For example, if you specify a format of "%05s," a Vuser ID of 5 is displayed as "00005," padding the single digit with four zeros. To pad the number with blank spaces, specify the number of spaces without a "0." For example, %4s adds blank spaces before the Vuser ID so that the resulting parameter string is 4 characters long.

You can specify a text string before and after the actual parameter value.

For example, if you specify a format of "Vuser No: %03s," then a Vuser ID of 1 is displayed as "**Vuser No: 001**."

You can add and delete formats for the following parameter types: Date/Time; Group Name; Iteration Number; Load Generator Name; Vuser ID.

To add a format to a parameter type:

1. In the Parameter Properties dialog box, select the parameter type that you want to format.
2. Enter the format symbols in the editable box and click **Add Format**.

Note: When you add a format to the list, VuGen saves it with the Vuser, making it available for future use.

To delete a format:

In the Parameter Properties dialog box, select an existing format from the list, and click **Delete format**.

To restore the original formats:

Click **Reset formats**.

Selecting an Update Method

When using several of the parameter types, VuGen lets you specify how to update the values for the parameters. To set an Update method, select a method from the **Update value on** list. The available update methods are:

- Each Occurrence
- Each Iteration
- Once

Each Occurrence

The **Each occurrence** method instructs the Vuser to use a new value for each occurrence of the parameter. 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

The **Each iteration** method instructs the Vuser to use a new value for each script 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. For more information about action blocks, see "Creating Action Blocks" on page 219.

Once

The **Once** method instructs the Vuser to update 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.

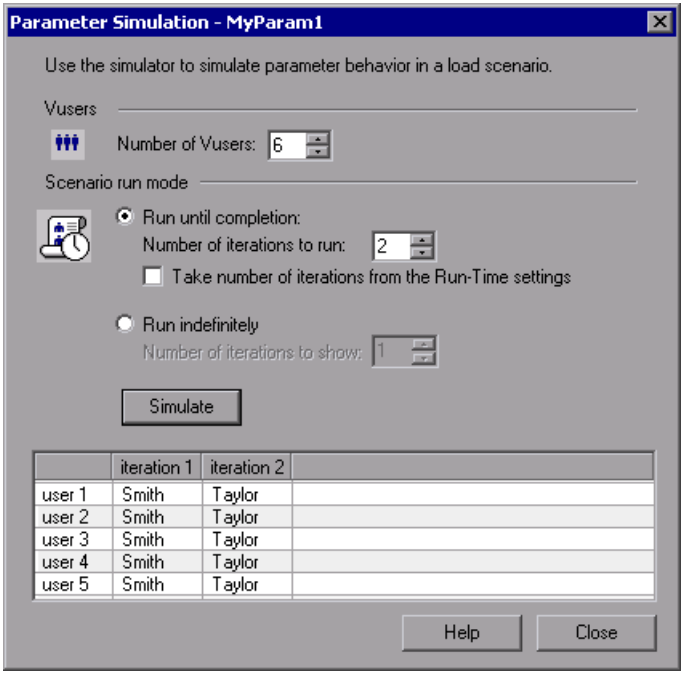
Simulating File Type Parameters

After you have created a File type parameter, you can use the File Parameter Simulator to simulate the parameter substitution in an actual scenario. This allows you to correct any wrong parameters before you run the script in the Controller. The File Parameter Simulator is only relevant for LoadRunner users.

Note: Not all types of Parameter Substitution can be simulated. If you choose **Select next row: Same line as...** or **Update value on: Each occurrence**, then the File Parameter Simulator will not open.

To run a File Parameter Simulation:

1 From the Parameter List dialog box, click **Simulate Parameter**. The Parameter Simulation dialog box opens.



2 Select the number of Vusers to run in the simulation from the **Number of Vusers** box.

3 Select Scenario run mode:

- ➤ If you choose **Run until completion**, select the number of iterations to run from the **Number of iterations to run** box, or take the number from the Run-Time settings.
- ➤ If you choose **Run indefinitely**, you must define how many iterations to show. The number of iteration in the Run-Time settings is ignored. The number you choose does not change the value distribution for each Vuser; it is only for viewing purposes.

Note:

- **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.

---

4 Click **Simulate**.

Note: The File Parameter Simulator can simulate up to 256 iterations and 256 Vusers.

Using the File Parameter Simulator

The following section illustrates how the File Parameter Simulator operates, demonstrating the difference between the Scenario run mode settings.

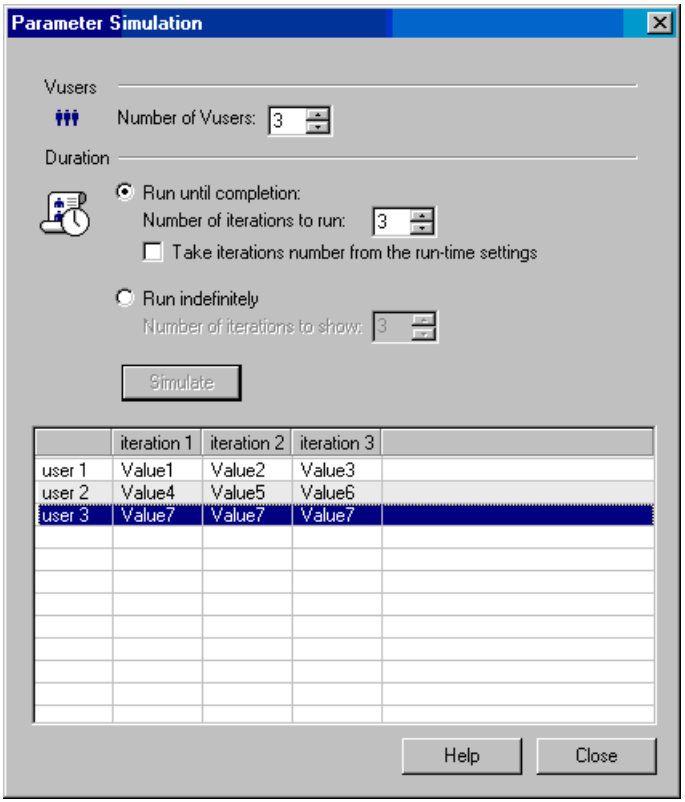
In the following examples, the settings in the Parameter List dialog box are:

- **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.



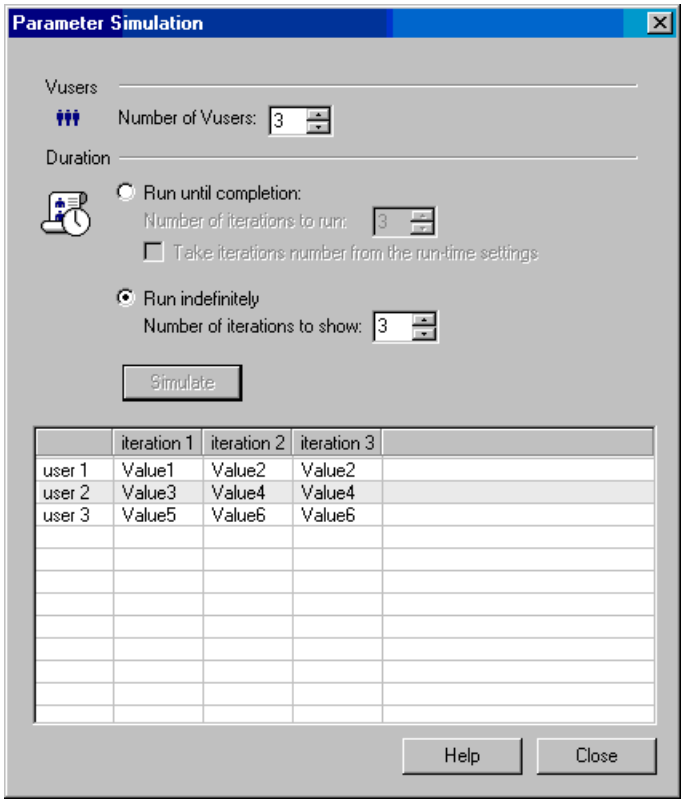
When the scenario run mode is set to **Run until completion**, the number of rows that each Vuser receives is the same as the number of iterations. The range allocation stops when there are no longer enough rows in the table.

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.

12

Correlating Statements

You can optimize Vuser scripts by correlating statements. VuGen's Correlated Query feature allows you to link statements by using the results of one statement as input for another.

The following information applies to all Vuser Scripts.

About Correlating Statements

The primary reasons for correlating statements are:

To simplify or optimize your code

For example, if you perform a series of dependent queries one after another, your code may become very long. To reduce the size of the code, you can nest the queries, but then you lose precision and the code becomes complex and difficult to understand. Correlating the statements enables you to link queries without nesting.

To generate dynamic data

Many applications and Web sites identify a session by the current date and time. If you try to replay a script, it will fail because the current time is different than the recorded time. Correlating the data enables you to save the dynamic data and use it throughout the scenario run.

To accommodate unique data records

Certain applications (for example databases) require the use of unique values. A value that was unique during recording is no longer unique for script execution. For example, suppose you record the process of opening a new bank account. Each new account is assigned a unique number which is unknown to the user. This account number is inserted into a table with a unique key constraint during recording. If you try to run the script as recorded, it tries to create an account with the recorded number, rather than a new unique number. An error will result because the account number already exists.

If you encounter an error when running your script, examine the script at the point where the error occurred. In many cases, a correlated query will solve the problem by enabling you to use the results of one statement as input to another.

The main steps in correlating a script are:

1. **Determine which value to correlate.**

   For most protocols, you can view the problematic statements in the Execution log. You double-click an error message and jump directly to its location.

   Alternatively, you can use the *WDiff* utility distributed with VuGen to determine the inconsistencies within your script. For more information, see "Comparing Vuser Scripts using WDiff" on page 212.

2. **Save the results.**

   You save the value of a query to a variable using the appropriate function. The correlating functions are protocol-specific. Correlation function names usually contain the string *save_param*, such as **web_reg_save_param** and **lrs_save_param**. Refer to the specific protocol chapters for an explanation on how to perform correlation. In several protocols, such as database and Web, VuGen automatically inserts the functions into your script.

3 Reference the saved values.

Replace the constants in the query or statement with the saved variables.

Several protocols have built-in automatic or partially automated correlation:

- For Java language Vusers, see “Correlating Java Scripts” in Volume II - the *Protocols user guide*.
- For Database Vusers, see “Correlating Database Vuser Scripts” in Volume II - the *Protocols user guide*.
- For Web Vusers, see “Setting Correlation Rules for Web Vuser Scripts” in Volume II - the *Protocols user guide*.
- For COM Vusers, see “Understanding COM Vuser Scripts” in Volume II - the *Protocols user guide*.

Using Correlation Functions for C Vusers

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. For similar functions for Java, Corba-Java, or RMI-Java Vusers, see “Using Correlation Functions for Java Vusers” on page 210.

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, refer to the *Online 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");
```

Using Correlation Functions for Java Vusers

To correlate statements for Java, CORBA-Java, and RMI-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-Java or RMI-Java script, VuGen performs correlation internally. For more information, see “Correlating Java Scripts” in Volume II - the *Protocols user guide*.

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
```

Comparing Vuser Scripts using WDiff

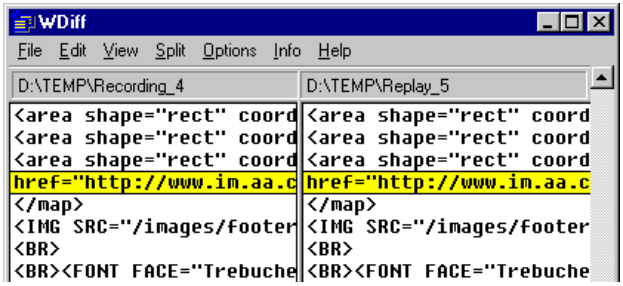
A useful tool in determining which values to correlate is *WDiff*. This utility lets you compare recorded scripts and results to determine which values need to be correlated.

If you are working with other protocols, you can view the Execution log to determine where the script failed and then use the *WDiff* utility to assist you in locating the values that need to be correlated.

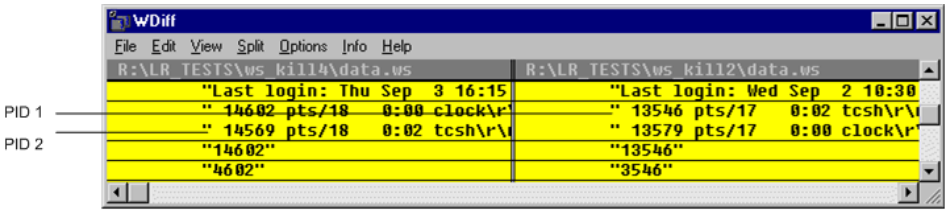
To use *WDiff* effectively, you record the identical operation twice, and compare the scripts (or data files for Tuxedo, WinSock, and Jolt). WDiff displays differences in yellow. Note that not all differences indicate a value to correlate. For example, certain receive buffers that indicate the time of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

To search for correlations using WDiff:

1. Record a script and save it.
2. Create a new script and record the identical operations. Save the script.
3. Select the section you want to compare (*Actions*, *data.ws*, and so forth).
4. Select **Tools** > **Compare with Vuser**. The Open Test box opens.
5. Specify a Vuser script for comparison (other than the one in the current VuGen window) and click **OK**. WDiff opens and the differences between the Vuser scripts are highlighted in yellow.



6 To display the differences only, double-click in the WDiff window.



7 Determine which values need to be correlated.

Note that in the above example, WDiff is comparing the *data.ws* from two Winsock Vuser scripts. In this instance, the value to be correlated is the PID for the *clock* processes, which differs between the two recordings.

Note: *WDiff* is the default utility, but you can specify a custom comparison tool. For more information, see "Comparison Tool" on page 34

To continue with correlation, refer to the appropriate section:

- For Java language Vusers, see "Correlating Java Scripts" in Volume II - the *Protocols user guide*.
- For Database Vusers, see "Correlating Database Vuser Scripts" in Volume II - the *Protocols user guide*.
- For Web Vusers, see "Setting Correlation Rules for Web Vuser Scripts" in Volume II - the *Protocols user guide*.
- For COM Vusers, see "Understanding COM Vuser Scripts" in Volume II - the *Protocols user guide*.
- For Tuxedo Vusers, see "Developing Tuxedo Vuser Scripts" in Volume II - the *Protocols user guide*.
- For WinSock Vusers, see "Working with Windows Socket Data" in Volume II - the *Protocols user guide*.

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"));
```

13

Configuring Run-Time Settings

After you record a Vuser script, you configure the run-time settings for the script. These settings specify how the script behaves when it runs.

The following information applies to all types of Vuser scripts except for GUI.

About Run-Time Settings

After you record a Vuser script, you can configure its run-time settings. The run-time settings define the way that the script runs. These settings are stored in the file *default.cfg,* located in the Vuser script directory. Run-time settings are applied to Vusers when you run a script using VuGen, the Controller, or Business Process Monitor.

Configuring run-time settings allows you to emulate different kinds of user activity. For example, you could 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 use the Run-Time Settings dialog box to display and configure the run-time settings tree. You can open these settings in one of the following ways:

- Click the **Run-Time Settings** button on the VuGen toolbar.
- Press the keyboard shortcut key **F4**.
- Choose **Vuser** > **Run-Time Settings**.

You can also modify the run-time settings from the LoadRunner Controller. For more information, refer to the product's documentation.

Note: For LoadRunner, the default run-time setting support the debugging environment of VuGen and the load testing environment of the Controller. The default settings are:

- **Think Time.** Off in VuGen and Replay as Recorded in the Controller.
- **Log.** Standard in VuGen and off in the Controller.
- **Download non-HTML resources.** Enabled in VuGen and the Controller.

The General run-time settings described in this chapter, apply to all types of Vuser scripts. They include:

- Run Logic (Iterations)
  - Pacing
  - Log
  - Think Time
  - Miscellaneous
  - Additional Attributes

For protocols that do NOT support multiple actions, such as WinSocket and Database (Oracle 2-tier, Sybase, MSSQL, and so on), the Iteration and Pacing options are both handled from the Pacing tab. Many protocols have additional run-time settings. For information about the specific run-time settings for these protocols, see the appropriate sections.

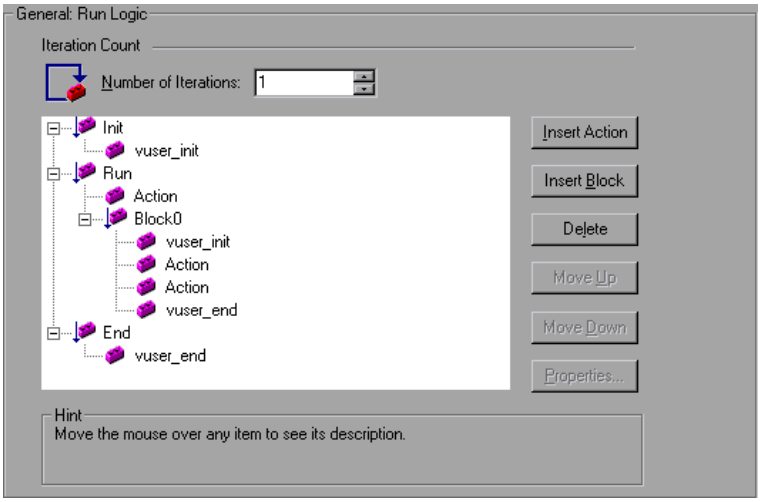
Configuring Run Logic Run-Time Settings (multi-action)

Note: The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 224.

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.

Open the Run-Time Settings and select the **General:Run Logic** node.



➤ **Number of Iterations.** The number of iterations. The Vusers repeat all of the Actions the specified number of times.

Note: For the LoadRunner Controller: If you specify a scenario duration in the Scheduling settings, they override the Vuser iteration settings. This means that if the duration is set to five minutes (the default setting), the Vusers will continue to run as many iterations as required for five minutes, even if the run-time settings specify only one iteration.

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

➤ **Action Blocks.** Action blocks are groups of actions within your script. You can set the properties of each block independently—its sequence, iterations, and weighting.

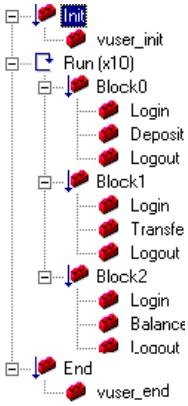
➤ **Sequence.** You can set the order of actions within your script. You can also indicate whether to perform actions sequentially or randomly.

- **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.

Creating Action Blocks

Action blocks are groups of actions within the Vuser script. You can create separate action blocks for groups of actions, adding the same action to several blocks. You can instruct VuGen to execute action blocks or individual actions sequentially or randomly. In the default sequential mode, the Vuser executes the blocks or actions in the order in which they appear in the iteration tree view.

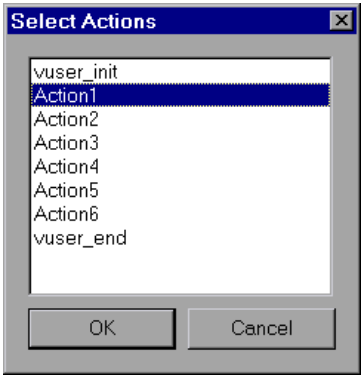
In the following example, *Block0* performs a deposit, *Block1* performs a transfer, and *Block2* submits a balance request. The *Login* and *Logout* actions are common to the three blocks.



You configure each block independently—its sequence and iterations.

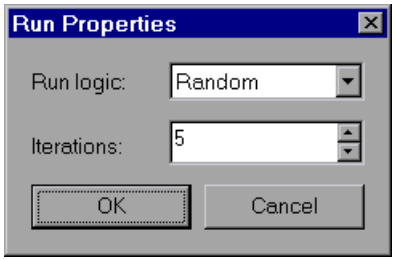
To configure actions and action blocks:

1. Create all of the desired actions through recording or programming.
2. Open the Run-Time setting. Select the **General:Run Logic** node.
3. Add a new action block. Click **Insert Block**. VuGen inserts a new Action block at the insertion point with the next available index (*Block0*, *Block1*, *Block2*).
4. Add actions to the block. Click **Insert Action**. The Select Actions list opens.



5. Select an action to add to the block and click **OK**. VuGen inserts a new action into the current block or section.
6. Repeat step 3 for each action you want to add to the block.
7. To remove an action or an action block, select it and click **Delete**.
8. Click **Move Up** or **Move Down** to modify an item's position.

9 Click **Properties** to set the number of iterations and run logic of the actions. The Run Properties dialog opens.

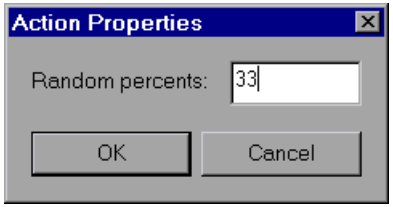


10 Select *Sequential* or *Random* from the **Run Logic** list, indicating to VuGen whether to run the actions sequentially or randomly.

11 Specify the number of iterations in the **Iterations** box. Note that if you define parameters within the action block, and you instruct VuGen to update their values each iteration, it refers to the global iteration—not the individual block iteration.

12 Click **OK**.

13 For blocks with Random run logic, set the weighting of each action. Right-click an action and choose **Properties**. The Action Properties dialog opens.



Specify the desired percent for the selected block or action. In the **Random Percents** box, specify a percentage for the current action. The sum of all percentages must equal 100.

14 Repeat the above steps for each element whose properties you want to set.

Pacing Run-Time Settings

Note: The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 224.

The Pacing Run-Time settings let you control the time between iterations. The pace tells the Vuser how long to wait between iterations of your actions. You instruct the Vusers to start each iteration using one of the following methods:

- **As soon as the previous iteration ends.** The new iteration begins as soon as possible after the previous iteration ends.
- **After the previous iteration ends with a fixed or random delay of …** 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. For example, you can specify to begin a new iteration at any time between 60 and 90 seconds after the previous iteration ends.

  When you run the script, VuGen shows the time the Vuser waited between the end of one iteration and the start of the next one, in the Execution Log.
- **At fixed or random intervals, every … [to …] seconds.** You specify the time between iteration—either a fixed number of seconds or a range of seconds from the beginning of the previous iteration. For example, you can specify to begin a new iteration every 30 seconds, or at a random rate ranging from 30 to 45 seconds from the beginning of the previous iteration. Each scheduled iterations will only begin when the previous iteration is complete.

  Each scheduled iteration will only begin when the previous iteration is complete. When you run the script, VuGen shows the time the Vuser waited between the end of one iteration and the start of the next one, in the Execution Log.

For example, assume that you specify to start a new iteration every four seconds:

- If the first iteration takes three seconds, the Vuser waits one second.
- If the first iteration takes two seconds to complete, the Vuser waits two seconds.
- If the first iteration takes 8 seconds to complete, the second iteration will start 8 seconds after the first iteration began. VuGen displays a message in the Execution Log to indicate that the iteration pacing could not be achieved.

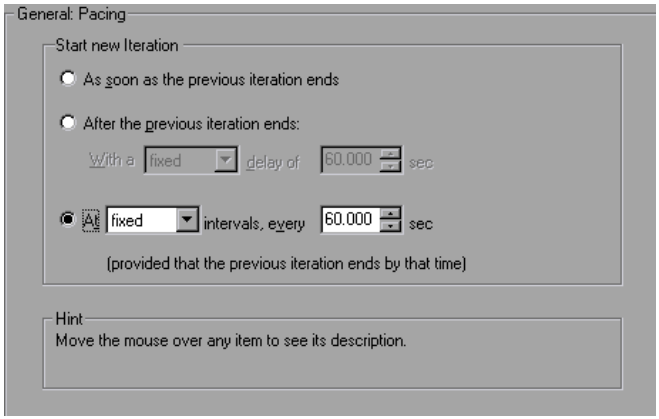
For further instructions about setting the Pacing options, see "Configuring Pacing Run-Time Settings (multi-action)" on page 223.

Configuring Pacing Run-Time Settings (multi-action)

You use the Pacing options to pace your actions by setting the time intervals between iterations.

To set the pacing between iterations:

1 Open the Run-Time Settings and select the **General:Pacing** node.



2 In the **Start New Iteration** section, select one of the following options:

- ➤ As soon as the previous iteration ends
- ➤ After the previous iteration ends
- ➤ At fixed or random intervals

3 For the **After the previous iteration ends** option:

- ➤ Select a delay type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random delay.

4 For the **At … intervals** option:

- ➤ Select a interval type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random interval.

5 Click **OK**.

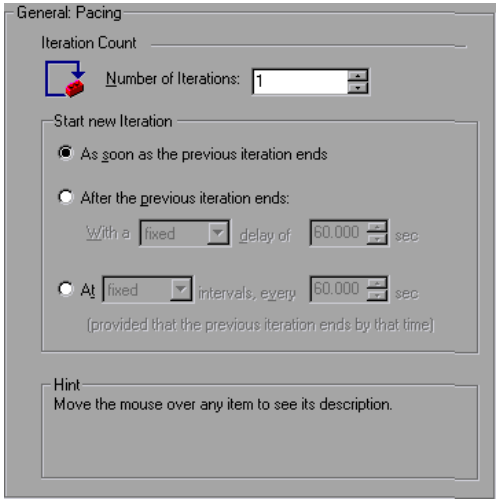
Setting Pacing and Run Logic Options (single action)

Note: The following section only applies to protocols that work with single actions—not multiple actions. If there is a **Pacing** node and not a **Run Logic** node under the General run-time settings, it is a single action protocol.

You can instruct a Vuser to repeat the *Action* 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.

To set the iteration and pacing preferences:

1 Click the **Run-Time Settings** button on the VuGen toolbar or select **Vuser** > **Run-Time Settings**. Click the **Pacing** node to display the iteration and pacing options.



2 Specify the number of iterations in the **Iteration Count** box. The Vuser repeats all of the Actions the specified number of times.

3 In the **Start New Iteration** section, select one of the following options:

- ➤ As soon as the previous iteration ends
- ➤ After the previous iteration ends
- ➤ At fixed or random intervals

4 For the **After the previous iteration ends** option:

- ➤ Select a delay type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random delay.

5 For the **At … intervals** option:

- ➤ Select a interval type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random interval.

6 Click **OK**.

For an overview of the pacing options, see “Pacing Run-Time Settings” on page 222.

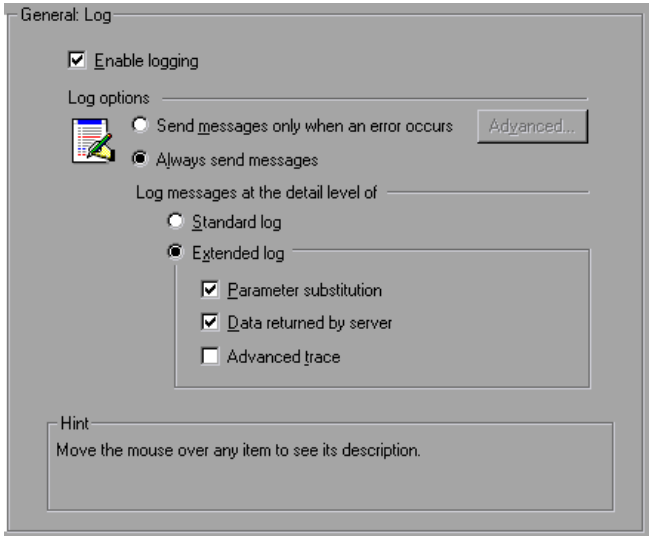
Configuring the Log Run-Time Settings

During execution, Vusers log information about themselves and their communication with the server. In a Windows environment, this information is stored in a file called *output.txt* in the script directory. In UNIX environments, the information is directed to the standard output. The log information is useful for debugging purposes.

The Log run-time settings let you determine how much information is logged to the output. You can select **Standard** or **Extended** log, or you can disable logging completely. Disabling the log is useful when working with many Vusers. If you have tens or hundreds of Vusers logging their run-time information to disk, the system may work slower than normal. During development, enable logging so that you will have information about the replay. You should only disable logging after verifying that the script is functional.

Note: You can program a Vuser script to send messages to an output log by using the **lr_error_message** and **lr_output_message** functions.

Click the **Run-Time Settings** button on select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Log** node to display the log options.



Enable Logging

This option enables automatic logging during replay—VuGen writes log messages that you can view in the Execution log. This option 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

The Log run-time settings allows you to adjust the logging level depending on your development stage.

You can indicate when to send log messages to the log: **Send messages only when an error occurs** or **Always send messages**. During development, you can enable all logging. Once you debug your script and verify that it is functional, you can enable logging for errors only.

If you choose to send messages only when errors occur, also known as JIT, (Just in Time) messaging, you can set an advanced option, indicating the size of the log cache. See "Setting the Log Cache Size" on page 229.

Setting the Log Detail Level

You can specify the type of information that is logged, or you can disable logging altogether.

Note: If you set **Error Handling** to "Continue on error" in the **General Run-Time Settings** folder, error messages are still sent to the Output window.

If you modify the script's Log Detail Level, the behavior of the **lr_message**, **lr_output_message**, and **lr_log_message** functions will not change—they will continue to send messages.

- **Standard Log**. Creates a standard log of functions and messages sent during script execution to use for debugging. Disable this option for large load testing scenarios or profiles.

  If the logging level is set to **Standard**, the logging mode is automatically set to **JIT logging** when adding it to a scenario or profile. If, however, the logging mode was disabled or set to **Extended**, then adding the script to a scenario or profile will not affect its logging settings.

- **Extended Log.** Creates an extended log, including warnings and other messages. Disable this option for large load testing scenarios or profiles.

  You can specify which additional information should be added to the extended log using the Extended log options:

  - **Parameter substitution.** Select this option to log all parameters assigned to the script along with their values. For more information on parameters, see Chapter 9, "Working with VuGen Parameters."
  - **Data returned by server.** Select this option to log all of the data returned by the server.

➤ **Advanced trace.** Select this option to log all of the functions and messages sent by the Vuser during the session. This option is useful when you debug a Vuser script.

The degree to which VuGen logs events (Standard, Parameter substitution, and so forth) is also known as the *message class*. There are five message classes: Brief, Extended, Parameters, Result Data, and Full Trace.

You can manually set the message class within your script using the **lr_set_debug_message** function. This is useful if you to want to receive debug information about a small section of the script only.

For example, suppose you set Log run-time settings to Standard log and you want to get an Extended log for a specific section of the script. You would then use the **lr_set_debug_message** function to set the Extended message class at the desired point in your script. You must call the function again to specify what type of extended mode (Parameter, Result Data, or Full Trace). Return to the Standard log mode by calling **lr_set_debug_message**, specifying Brief mode. For more information about setting the message class, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Setting the Log Cache Size

The Advanced options for the Log Run-Time settings, let you indicate the size of the log cache. The log cache stores raw data about the test execution, to make it available should an error occur. When the contents of the cache exceed the specified size, it deletes the oldest items. The default size is 1KB.

The following is the sequence of the logging:

1. You indicate to VuGen to log messages only when an error occurs, by selecting **Send messages only when an error occurs**.
2. VuGen stores information about the test execution in the log cache without writing it to a file. If this information exceeds 1 KB, it overwrites the oldest data. The Execution Log tab also remains empty, since it is a dump of the log file's contents.
3. When an error occurs (either an internal error or a programmed error using **lr_error_message**), VuGen places the contents of the cache into the log file and Execution Log tab. This allows you to see the events that led up to the error.

When an error occurs and VuGen dumps its stored cache into the log file, the actual file size will be greater than the cache size. For example, if your cache size is 1KB, the log file size may be 50 KB. This is normal and only reflects the overhead required for formatting the raw data into meaningful sentences.

Note that in JIT mode, the output of **lr_message** and **lr_log_message**, are only sent to the Output window or log file, if their output was in the log cache at the time of the error. Check the Execution Log for the specified message strings.

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.

Configuring the Think Time Settings

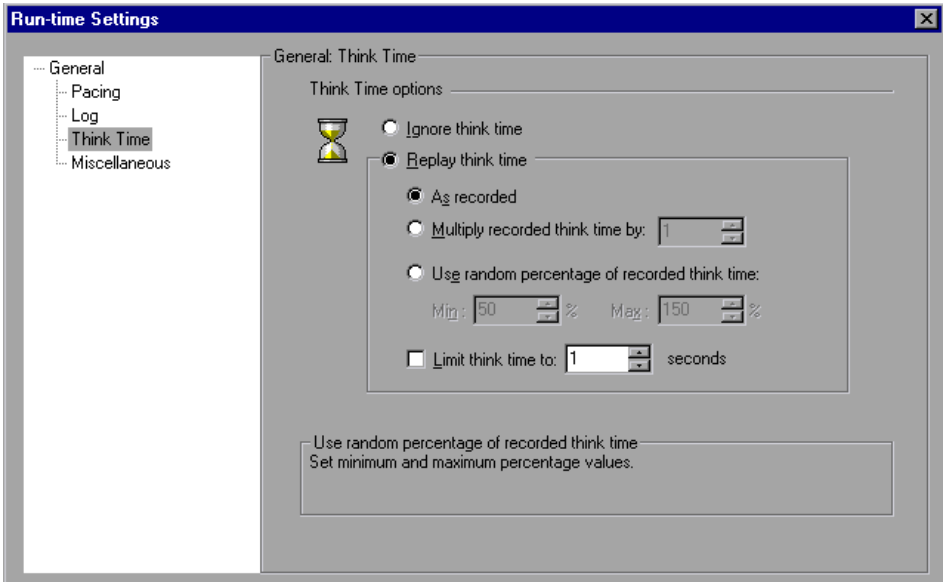
Vuser *think time* emulates the time that a real user waits between actions. For example, when a user receives data from a server, the user may wait several seconds to review the data before responding. This delay is known as the *think time*. VuGen uses **lr_think_time** functions to record think time values into your Vuser scripts. The following recorded function indicates that the user waited 8 seconds before performing the next action:

```
lr_think_time(8);
```

When you run the Vuser script and the Vuser encounters the above **lr_think_time** statement, by default, the Vuser waits 8 seconds before performing the next action. You can use the Think Time run-time settings to influence how the Vuser uses the recorded think time when you run the script.

For more information about the **lr_think_time** function and how to modify it manually, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Click the **Run-Time Settings** button on the VuGen toolbar or select **Vuser** > **Run-Time Settings**. Select the **General:Think Time** node to display the Think Time options:



Think Time Options

By default, when you run a Vuser script, the Vuser uses the think time values that were recorded into the script during the recording session. VuGen allows you to use the recorded think time, ignore it, or use a value related to the recorded time:

- **Ignore think time.** Ignore the recorded think time—replay the script ignoring all **lr_think_time** functions.
- **Replay the think time.** The second set of think times options let you use the recorded think time:
  - **As recorded.** During replay, use the argument that appears in the lr_think_time function. For example, lr_think_time(10) waits ten seconds.

- **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.
- **Use random percentage of the 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%).
- **Limit think time to.** Limit the think time's maximum value.

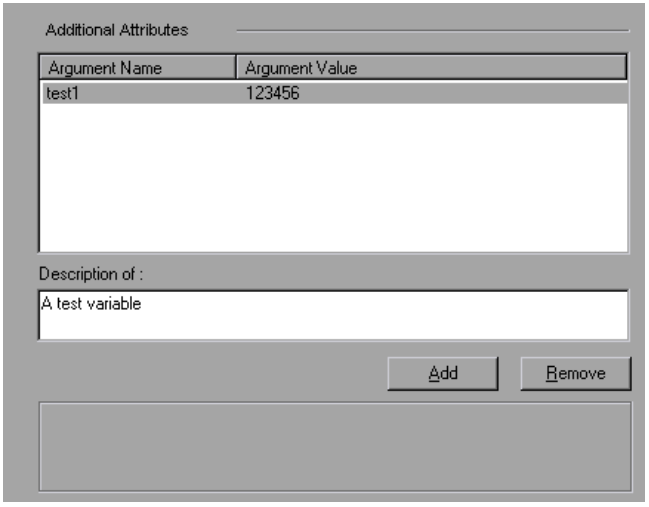
Configuring Additional Attributes Run-Time Settings

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 set additional attributes:

1 Click the Run-Time Settings button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Additional Attributes** node from the tree in the left pane.



2 Click **Add** to add a new command line argument entry. Enter the argument name and its value.

3 Click **Remove** to remove the selected argument.

Configuring Miscellaneous Run-Time Settings

You can set the following Miscellaneous run-time options for a Vuser script: Note that the Multithreading and Automatic Transaction options are not applicable to HP Business Availability Center.

- Error Handling
- Multithreading
- Automatic Transactions

Click the **Run-Time Settings button** or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Miscellaneous** node from the tree in the left pane.

The *Miscellaneous* settings apply to all Vuser script types.

Error Handling

- **Continue on Error.** This setting instructs Vusers to continue script execution when an error occurs. This option is turned off by default, indicating that the Vuser will exit if an error occurs.
- **Fail open transactions on lr_error_message.** This option 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 programmed *If* statement, when a certain condition is met.
- **Generate Snapshot on Error.** This option 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.

It is not recommended to enable both the **Continue on Error** and **Generate Snapshot on Error** options in a load test environment. This configuration may adversely affect the Vusers' performance.

Error Handling for Database Vusers

When working with database protocols (LRD), you can control error handling for a specific segment of a script. To mark a segment, enclose it with LRD_ON_ERROR_CONTINUE and LRD_ON_ERROR_EXIT statements. The Vuser applies the new error setting to the whole segment. If you specify Continue on Error, VuGen issues a messages indicating that it encountered an error and is ignoring it.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
lrd_stmt(Csr1, "select…"…);
lrd_exec(…);
```

To instruct the Vuser to continue on error for the entire script except for a specific segment, select the Continue on Error option and enclose the segment with LRD_ON_ERROR_EXIT and LRD_ON_ERROR_CONTINUE statements:

```
LRD_ON_ERROR_EXIT;
lrd_stmt(Csr1, "select…"…);
lrd_exec(…);
LRD_ON_ERROR_CONTINUE;
```

In addition to the LRD_ON_ERROR statements, you can control error handling using *severity levels*. LRD_ON_ERROR statements detect all types of errors—database related, invalid parameters, and so on. If you want the Vuser to terminate only when a database operation error occurs (Error Code 2009), you can set a function's severity level. All functions that perform a database operation use severity levels, indicated by the function's final parameter, *miDBErrorSeverity*.

VuGen supports the following severity levels:

Definition	Meaning	Value
LRD_DB_ERROR_SEVERITY_ERROR	Terminate script execution upon database access errors. (default)	0
LRD_DB_ERROR_SEVERITY_WARNING	Continue script execution upon database access errors, but issue a warning.	1

For example, if the following database statement fails (e.g. the table does not exist), the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 0);
```

To instruct VuGen to continue script execution, even when a database operation error occurs, change the statement's severity level from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 1);
```

Note: When you enable Continue on Error, it overrides the “0” severity level; script execution continues even when database errors occur. However, if you disable Continue on Error, but you specify a severity level of “1”, script execution continues when database errors occur.

Error Handling for RTE Vusers

When working with RTE Vusers, you can control error handling for specific functions. You insert an **lr_continue_on_error(0);** statement before the function whose behavior you want to change. The Vuser uses the new setting until the end of the script execution or until another **lr_continue_on_error** statement is issued.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
TE_wait_sync();
TE_type(...);
```

To instruct the Vuser to continue on error for the entire script, except for the following segment, select the Continue on Error option and enclose the segment with lr_continue_on_error statements, using 0 to turn off Continue on Error and 1 to turn it back on:

```
lr_continue_on_error(0);
TE_wait_sync();
lr_continue_on_error(1);
....
```

Multithreading

Vusers support multithread environments. The primary advantage of a multithread environment is the ability to run more Vusers per load generator. Only threadsafe protocols should be run as threads. (not applicable to HP Business Availability Center)

Note: The following protocols are not threadsafe: Sybase-Ctlib, Sybase-Dblib, Informix, Tuxedo, and PeopleSoft-Tuxedo.

➤ To enable multithreading, click **Run Vuser as a thread**.

➤ To disable multithreading and run each Vuser as a separate process, click **Run Vuser as a process**.

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.

Automatic Transactions

You can instruct LoadRunner (not applicable to HP Business Availability Center) to handle every step or action in a Vuser script as a transaction. This is called using automatic transactions. LoadRunner assigns the step or action name as the name of the transaction. By default, automatic transactions per action are enabled.

- ➤ To disable automatic transactions per action, clear the **Define each action as a transaction** check box. (enabled by default)
- ➤ To enable automatic transactions per step, check the **Define each step as a transaction** check box. (disabled by default)

If you disable automatic transactions, you can still insert transactions manually during and after recording. For more information on manually inserting transactions, see Chapter 8, “Enhancing Vuser Scripts.”

Note: 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.

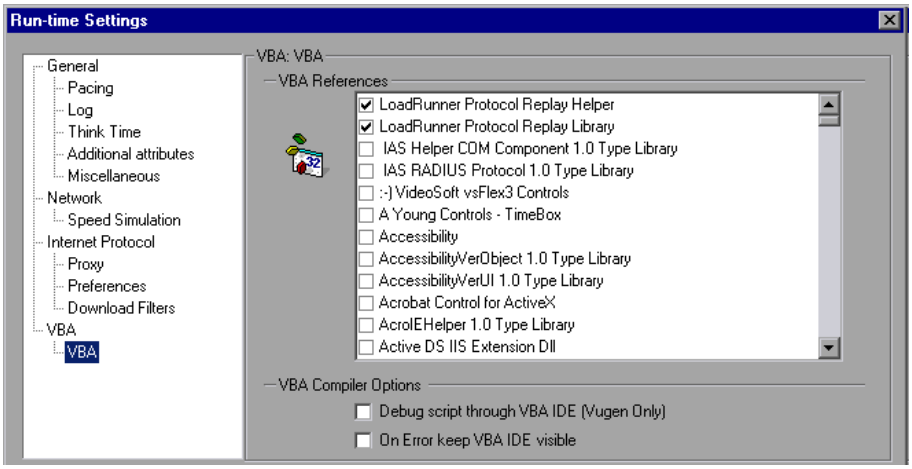
Setting the VB Run-Time Settings

Before running your Visual Basic script, you indicate which libraries to reference during replay. VuGen displays a list of all of the libraries stored on the machine.

You use the Run-Time Settings dialog box to display and configure the run-time settings. To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar.

To set the VBA Run-Time settings:

1 Open the **Run-Time Settings** dialog box and select the **VBA:VBA** node.



2 In the **VBA References** section, select the reference library that you want to use while running the script. Select a library to display its description and version in the bottom of the dialog box.

3 Select the appropriate compiler options:

Select **Debug script through VBA IDE** to enable debugging through the Visual Basic IDE (Integrated Development Environment).

Select **On Error keep VBA IDE visible** to keep the Visual Basic IDE visible during script execution.

4 Choose **OK** to apply the run-time settings.

14

Configuring Network Run-Time Settings

To simulate the speed over a network, you configure the Network run-time settings.

The following information applies to all Internet-related protocols, Citrix ICA, Oracle NCA, and WinSock.

For information about the general run-time settings that apply to all Vusers, see Chapter 13, “Configuring Run-Time Settings.”

About Network Run-Time Settings

After developing a Vuser script, you set the run-time settings. These settings let you configure your Internet environment so that Vusers can accurately emulate real users. You can set Internet-related run-time settings for Proxy, Browser, Speed Simulation, and other advanced preferences.

You set the run-time settings by opening the Run-Time Settings dialog box and selecting the appropriate node:

To display the Run-Time Settings dialog box:

➤ Click the **Run-Time Settings** button on the VuGen toolbar.

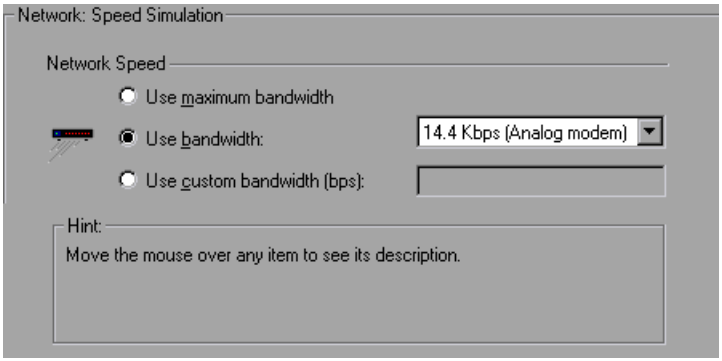
➤ Press the keyboard shortcut key **F4**.

➤ Choose **Vuser** > **Run-Time Settings**.

Note that you can also modify the run-time settings from the LoadRunner Controller. For more information, refer to your product's documentation.

Setting the Network Speed

You use the **Network:Speed Simulation** node in the Run-Time Settings tree, to set the modem emulation for your testing environment.



Speed Simulation

Using the Speed Simulation settings, you can select a bandwidth that best emulates the environment under test. The following options are available:

➤ **Use maximum bandwidth**. By default, bandwidth emulation is disabled and the Vusers run at the maximum bandwidth that is available over the network.

➤ **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.

15

Running Vuser Scripts in Stand-Alone Mode

After you develop a Vuser script and set its run-time settings, you test the Vuser script by running it in stand-alone mode.

The following information applies to all types of Vuser scripts.

About Running Vuser Scripts in Standalone Mode

After creating a script, you check its functionality by running it in standalone mode, directly from the VuGen interface. If the script is UNIX-based, you run it from a UNIX command line. To run GUI Vusers in standalone mode, use WinRunner.

When the standalone execution is successful, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, refer to the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Before you run a script in standalone mode, you can:

- enhance the script with Vuser functions (see Chapter 8, "Enhancing Vuser Scripts")
- parameterize the script (see Chapter 9, "Working with VuGen Parameters")
- correlate queries in the script (see Chapter 12, "Correlating Statements")

The above steps are optional and may not apply to all scripts.

Running a Vuser Script in VuGen

After developing a Vuser script, run it using VuGen to verify that it executes correctly. You can set several options for replay.

Note: VuGen runs Vuser scripts on Windows platforms only. To run UNIX-based Vuser scripts, see "Running a Vuser Script from a UNIX Command Line" on page 260.

Configuring Replay Options

You can run a Vuser script in animated mode or non-animated mode. When you run in animated mode, VuGen highlights the line of the Vuser script being executed at the current time. You can set a delay for this mode, allowing you to better view the effects of each step. When you run in non-animated mode, VuGen executes the Vuser script, but does not indicate the line being executed.

- **Animated run delay.** The time delay in milliseconds between commands. The default delay value is 0.
- **Only animate functions in Actions sections.** Only animates the content of the Action sections, but not the *init* or *end* sections.
- **Prompt for results directory.** Prompts you for a results directory before running a script from VuGen. If this option is not selected, VuGen automatically names the directory *result1*. Subsequent script executions will automatically overwrite previous ones unless you specify a different result file. Note that results are stored in a subdirectory of the script.
- **After replay show.** Instructs VuGen how to proceed after the replay:
  - **View before replay.** Return to the view you had before replay.
  - **Replay summary.** Go directly to the Replay Summary window in the Workflow Wizard.
  - **Visual Test Results**. Open the Test Results Summary. (This is the same as choosing **View** > **Test Results** after replay.)

Use Defaults

When you click **Use Defaults**, VuGen resets the original values:

Animated run delay to 0 msec.

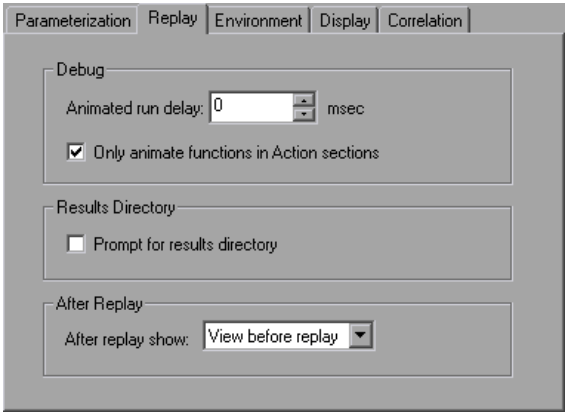
Only animate functions in action sections becomes enabled.

Prompt for results directory becomes disabled.

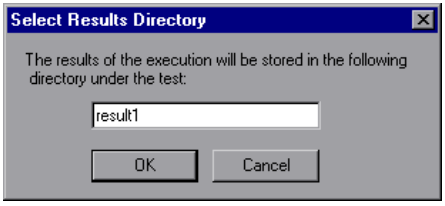
After replay show is set to View before replay.

To enable animation and set its properties:

1. Select **View** > **Animated Run** to run in animated mode. VuGen places a check mark beside the **Animated Run** menu option to enable animated mode.
2. To set the delay for the animated run, select **Tools** > **General Options**. The General Options dialog box opens.



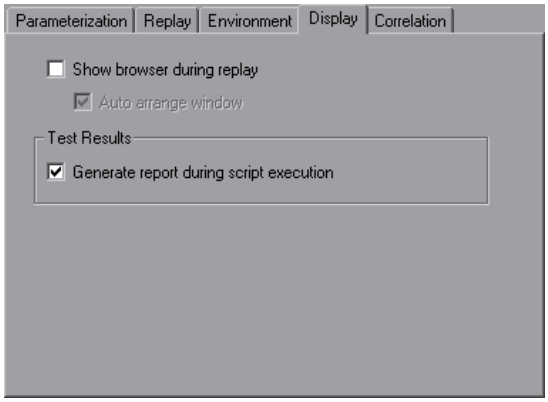
3. Select the **Replay** tab.
4. In the **Animated run delay** box, specify a delay in milliseconds and click **OK**.
5. Select **Only animate functions in Actions sections** to animate only the content of the Action sections.
6. Select **Prompt for results directory** to be prompted for a results directory before running a script from VuGen. The Select Results Directory dialog box opens when you click the run command.



7. Type a directory name for the execution results, or accept the default name and click **OK**.

Setting the Display Options

If you are running a Web Vuser script, you can set the Display options (**Tools** > **General Options**). These options specify whether to display VuGen's run-time viewer, whether to generate a report during script execution, and so forth.



- **Show browser during replay.** Enables the run-time viewer. The **Auto arrange window** options instructs VuGen to minimize the run-time viewer when script execution is complete.
- **Generate report during script execution.** Instructs a Vuser to generate a Results Summary report. You can open the report after script execution by selecting **View** > **Test Results**.

By default, the **Show browser during replay** option is disabled, and the **Generate report during script execution** option is enabled. To restore these values, click **Use Defaults**.

For more information on how to use these options for debugging, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 256.

To set the Display options:

1. Select **Tools** > **General Options** from the VuGen menu. The General Options dialog box opens. Select the **Display** tab.

2. Select **Show browser during replay** to enable the run-time viewer. Select **Auto arrange window** to minimize the run-time viewer when script execution is complete.
3. Select **Generate report during script execution** to instruct a Vuser to generate a Results Summary report. You can open the report after script execution by selecting **View** > **Test Results**.
4. Click **OK** to accept the settings and close the General Options dialog box.

Replaying a Vuser Script

Before you integrate a script into a test or production environment, you run it from VuGen to make sure it is functional. VuGen provides several tools that allow you to monitor the replay and locate any existing and potential problems. These include:

- Viewing the Replay Log
- The Run-Time Data Tab
- The Run Step by Step Command
- Breakpoints
- Bookmarks
- Go To Commands

To replay a script in VuGen:

1. Select **Vuser** > **Run**.

   The Output window opens at the bottom of the VuGen main window—or clears, if already open—and VuGen begins executing the Vuser script. In tree view, VuGen runs the Vuser script from the first icon in the script. In Script view, it runs the Vuser script from the first line of the script.

2. Click the Output window's **Replay Log** tab for a log of all of the actions of the Vuser, along with warnings and errors. For more information, see "Viewing the Replay Log" on page 249.

3. To view a summary of the run-time data and the parameters as they are being used, see the Output window's **RunTime Data** tab. For more information, see "The Run-Time Data Tab" on page 250.
4. To hide the Output window during or after a script run, select **View** > **Output Window**. VuGen closes the Output window and removes the check mark from next to **Output Window** on the **View** menu.
5. To interrupt a Vuser script that is running, select **Vuser** > **Pause**, to temporarily pause the script run, or **Vuser** > **Stop**, to end the script run.

Viewing the Replay Log

The Output window's Replay Log displays messages that describe the actions of the Vuser as it runs. This information tells you how the script will run when executed in a scenario or profile.

When script execution is complete, you examine the messages in the Replay Log to see whether your script ran without errors.

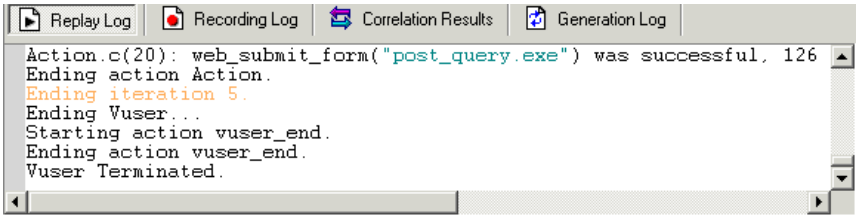
Various colors of text are used in the Replay Log.

- **Black.** Standard output messages.
- **Red.** Standard error messages.
- **Green.** Literal strings, such as URLs, that appear between quotation marks.
- **Blue.** Transaction Information (starting, ending, status and duration).
- **Orange.** The beginning and ending of iterations.

If you double-click on a line beginning with the Action name, the cursor jumps to the step within the script that generated.

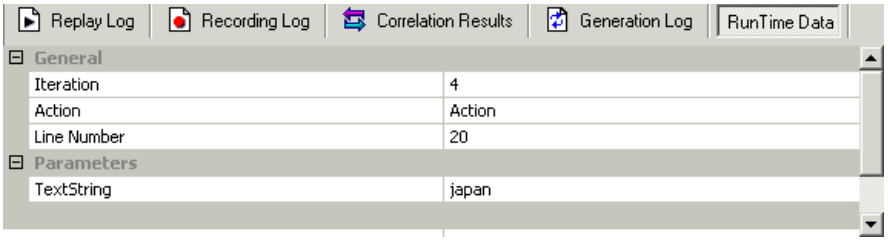
For more information on closing and opening the Output window, see "Replaying a Vuser Script" on page 248.

The following example shows Replay Log messages from a Web Vuser script run.



The Run-Time Data Tab

You can track the script information that becomes updates during replay, using the Run Time Data tab.



During replay, click the rightmost tab, **RunTime Data**. The tab contains two expandable/collapsible sections:

- **General.** The general section shows the current iteration number, the Action name of the currently replayed step, and the line number within the script (Script view).
- **Parameters.** The parameters section shows all parameters defined 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. For more information, see Chapter 9, “Working with VuGen Parameters.”

Note that the RunTime Data tab is not accessible after the test run, since it only displays data that changes during replay.

Using VuGen's Debugging Features

VuGen contains two options to help debug Vuser scripts—the Run Step by Step command and breakpoints. These options are not available for VBscript and VB Application type Vusers.

VuGen contains additional features to help debug Web Vuser scripts. For details, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 256.

To view the Debug toolbar:

Right-click the toolbar area and select **Debug**. The Debug toolbar appears in the toolbar area.

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.

To run the script step by step:

1. Select **Vuser** > **Run Step by Step**, or click the **Step** button on the Debug toolbar.

   VuGen executes the first line of the script.

2. Continue script execution by clicking the **Step** button until the script run completes.

Breakpoints

Breakpoints pause execution at specific points in the script. This enables you to examine the effects of the script on your application at pre-determined points during execution. To manage the breakpoints, use the Breakpoint Manager.

To set breakpoints:

1. Place the cursor on the line in the script at which you want execution to stop.

2. Select **Insert** > **Toggle Breakpoint**, or click the **Breakpoint** button in the Debug toolbar. Alternatively, press F9 on the keyboard. The Breakpoint symbol () appears in the left margin of the script.

3. To disable a breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Enable / Disable Breakpoint** button on the Debug toolbar. A white dot inside the Breakpoint symbol indicates a disabled breakpoint. When one breakpoint is disabled, script execution is paused at the following breakpoint. Click the button again to enable the breakpoint.

   To remove the breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Breakpoint** button or press F9.

To run the script with breakpoints:

1. Begin running the script as you normally would.

   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.

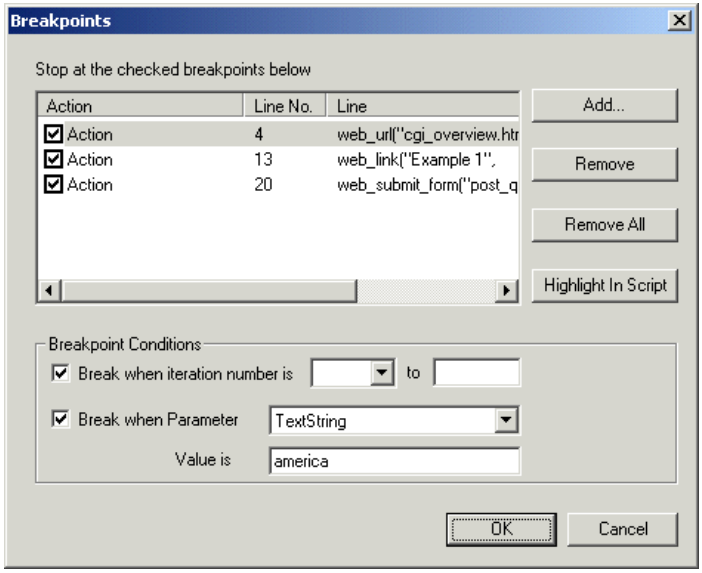
2. To resume execution, select **Vuser** > **Run**.

   Once restarted, the script continues until the next breakpoint is encountered or until the script is completed.

The Breakpoint Manager

You can view and manage breakpoints using the Breakpoint Manager. From the Breakpoint Manager you can manipulate all of the breakpoints in your script.

To open the Breakpoint Manager, select **Edit** > **Breakpoints**.



To jump to the breakpoint location in the script:

1. Select a breakpoint from the list.
2. Click **Highlight in Script**. The line in the script becomes highlighted.

   Note that you can only highlight one breakpoint at a time.

Managing Breakpoints

From the Breakpoint Manager, you can add, remove, disable, or conditionalize a breakpoint.

To add a breakpoint:

1. Click **Add**. The Add Breakpoint dialog box opens.
2. Select an **Action** and specify the **Line** number where you want add the breakpoint.
3. Click **OK.** The Breakpoint is added to the list of breakpoints.

To remove a breakpoint:

1. To remove a single breakpoint, select the breakpoint and click **Remove**.
2. To remove all the breakpoints at once, click **Remove All**.

To enable/disable a breakpoint:

1. To enable a breakpoint, in the Action column, select the action's check box.
2. To disable a breakpoint, in the Action column, clear the action's check box.

From the Breakpoint Manager, you can set breakpoints to pause execution under certain conditions.

To conditionalize a breakpoint:

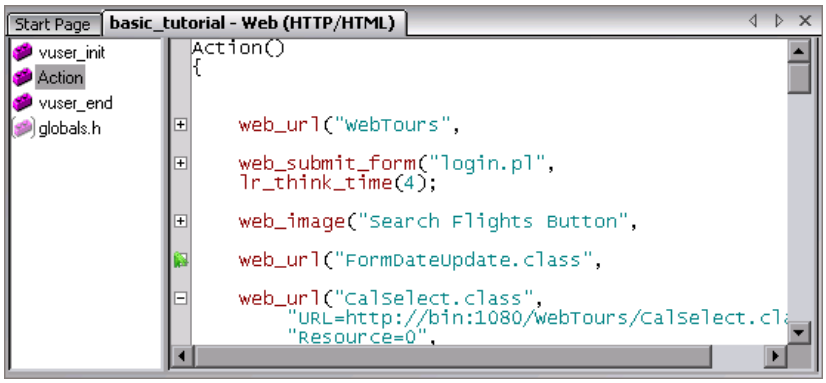
1. To pause the script after a specific number of iterations, select **Break when iteration number is** and enter the desired number.
2. To pause the script when parameter *X* has a specific value, select **Break when Parameter** *X* **Value is** and enter the desired value. For more information about parameters, see Chapter 9, "Working with VuGen Parameters."

Bookmarks

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

To create a bookmark:

1. Place the cursor at the desired location and press Ctrl + F2. VuGen places an icon in the left margin of the script.



2. To remove a bookmark, click on the desired bookmark and press Ctrl + F2. VuGen removes the bookmark icon from the left margin.

3. To move between bookmarks:

   To move to the next bookmark, click F2.

   To navigate to the previous bookmark, click Shift + F2.

You can also create and navigate between bookmarks through the **Edit** > **Bookmarks** menu item.

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

Go To Commands

To navigate around the script without using bookmarks, you can use the Go To command. Choose **Edit** > **Go To Line** and specify the line number of the script. This navigation is also supported in Tree view.

If you want to examine the Replay log messages for a specific step or function, select the step in VuGen and choose **Edit** > **Go To Step in Replay Log.** VuGen places the cursor at the corresponding step in the Output window's Replay Log tab.

Using VuGen's Debugging Features for Web Vuser Scripts

VuGen provides two additional tools to help you debug Web Vuser scripts—the run-time viewer (online browser) and the Results Summary report.

- You can instruct VuGen to display a run-time viewer when you run a Web Vuser script. The run-time viewer was developed specifically for use with VuGen—it is unrelated to the browser that you use to record your Vuser scripts. The run-time viewer shows each Web page as it is accessed by the Vuser. This is useful when you debug Web Vuser scripts because it allows you to check that the Vuser accesses the correct Web pages.
- 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, choose **View** > **Test Results** and click F1 to open the online help or refer to "Viewing Test Results" in Volume II - the *Protocols user guide*.

For information on setting the above Display options, see "Setting the Display Options" on page 247.

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.

Working with VuGen Windows

You can show and rearrange VuGen's windows to view the relevant data for your script, using the following features:

- **Show/Hide the Output Window.** Select **View** > **Output Window** to show and hide the Output window below the VuGen script editor. The Output window has several tabs, depending on the protocol. The most common tabs are the Replay Log, Recording Log, Generation Log, and Correlation Results. For more information, see "Viewing the Replay Log" on page 249.
- **Display All Thumbnails.** Select **View** > **Show All Thumbnails** to show all of the script's steps as thumbnails. To show thumbnails for primary steps only, clear this option. For more information, see "Viewing Script Thumbnails" on page 43.
- **Display Grids.** Select **View** > **Enable Data Grids** to enable grid display of the data in the protocols that support data grids (Database, COM, and Microsoft . NET). The grids appear inside the script.
- **Window Manipulation.** Select **Window** > **Close All** to close all of the open scripts. If necessary, VuGen will prompt you to save the unsaved scripts.

Find In Files

You use Find In Files to search for any string or expression in all the files of the script you currently have open.

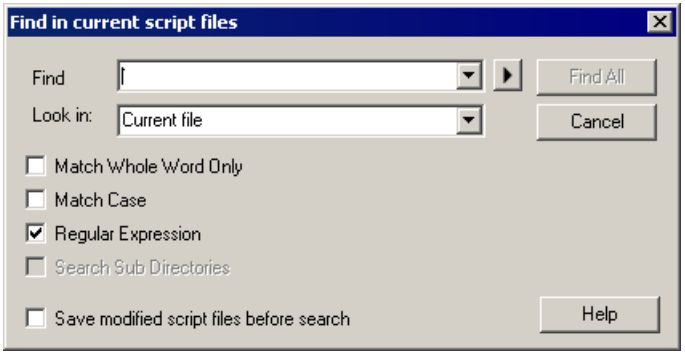
For example, if VuGen fails to replay a script due to an error in the replay log, you can search through all the files in the script simultaneously for a specific value that you think may be causing the failure.

Vugen displays all matches in a separate results window. You double-click on any line in the window to open the relevant file.

Note: VuGen also includes a regular **Find** feature. With this feature, you can search for values in only one file at a time. The value is highlighted in the script itself, and you press F3 to move to the next match.

To search with Find In Files:

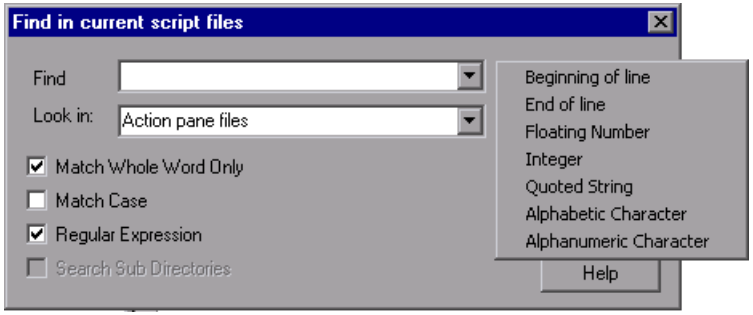
1. Open a script in VuGen.
2. Select **Edit** > **Find in current script files.** The Find in current script files dialog box opens.



3. In the **Find** box, enter the string you want to search for. You can select any of the previous ten searches from the drop-down list.

You can also search with a regular expression. A regular expression is a search string made up of a single character, or a set of characters, that is interpreted by VuGen as a pattern in the script. VuGen searches the script for values that match that pattern.

If you select **Regular Expression**, the arrow button by the **Find** box is activated, and provides seven common regular expressions that you can select instead of typing in manually.



4 In the **Look in** box, specify the script directory in which to search. You can also select from the two options provided by VuGen in the drop-down list.

- **Current file.** Searches the file currently displayed in the right pane.
- **Action pane files.** Searches all files that appear in the left pane.

If you specify a script directory other than those provided, the **Search Sub Directories** option is activated. You select this option to expand the search to sub directories of the current Vuser folder.

5 Select the desired settings from: **Match Whole Word Only** and **Match Case**.

6 Click **Find All**. The Search Results tab opens.

Note: It is recommended that you save the script before you search. If you make changes in the results, you might not be able to return to the original script. Note that Find In Files only searches the last saved version of the script. The **Save modified script files before search** option appears if you have already made changes to the script.

Running a Vuser Script from a Command Prompt

You can test a Vuser script from a Command Prompt or from the Windows Run dialog box—without the VuGen user interface.

To run a script from a DOS 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.

Running a Vuser Script from a UNIX Command Line

When using VuGen to develop UNIX-based Vusers, you must check that the recorded script runs on the UNIX platform. To verify that your script runs correctly, follow these steps:

1. **Test the recorded script from VuGen.**

   Run the recorded script from VuGen to check that the script runs correctly on a Windows-based system.

2. **Copy the Vuser script files to a UNIX drive.**

   Transfer the files to a local UNIX drive.

3. **Check the Vuser setup on the UNIX machine by using verify_generator.**

   For details, see “The verify_generator Test” on page 261.

4. **Test the script from the UNIX command line.**

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

```
run_db_vuser.sh script_name.usr
```

The verify_generator Test

The verify utility checks the local host for its communication parameters and its compatibility with all types of Vusers.

The utility 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

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

```
verify_generator
```

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]
```

Command Line Options: run_db_vuser Shell Script

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

--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 directory. For example, the driver for CtLib located in the /bin directory, 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 directory.

By default, *run_db_vuser.sh* runs **cpp**, **cci**, and **execute** in verbose mode. It uses the driver in the *VuGen installation*/bin directory, and saves the results to an output file in the Vuser script directory. You must always specify a *.usr* file. If you are not in the script directory, 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 directory called results1. The results directory must be an existing directory—it will not be created automatically:

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

Integrating Scripts into Tests

Once you have successfully run a script in standalone mode to verify that it is functional, you integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile.

When you integrate a test, you provide information indicating:

- which script
- Vusers that will run the script
- load generator upon which the script will be executed
- scheduling

For more information, refer to the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

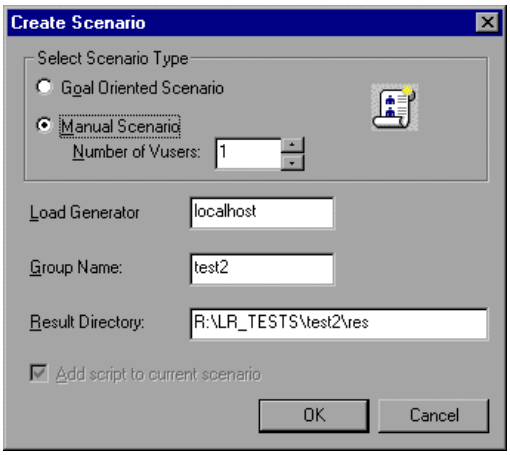
Using VuGen to Create a LoadRunner Scenario

Note: The following section only applies to LoadRunner. For information on integrating scripts into Business Process profiles, refer to the *HP Business Availability Center* 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 a scenario from VuGen:

1 Choose **Tools** > **Create Controller Scenario**. The Create Scenario dialog box opens.



2 Choose either a goal oriented or a manual scenario.

In a goal-oriented scenario, LoadRunner automatically builds a scenario based on the goals you specify, whereas in a manual scenario, you specify the number of Vusers to run.

3 For a manual scenario, enter the number of Vusers you want to execute the script.

4 Enter the name of the machine upon which you want the Vusers to run, in the **Load Generator** box.

5 For a manual scenario, users with common traits are organized into groups. Specify a new group name for the Vusers in the **Group Name** box.

6 For a goal-oriented scenario, specify a **Script Name**.

7 Enter the desired location for the results in the **Result Directory** box.

8 If a scenario is currently open in the Controller and you want to add the script to this scenario, select the **Add script to current scenario** check box. If you clear the check box, LoadRunner opens a new scenario with the specified number of Vusers.

9 Click **OK**. VuGen opens the Controller in the Vuser view.

10 If you configured the Controller to save the script on a shared network drive, you may need to perform path translation.

For more information, refer to the *HP LoadRunner Controller User's Guide*.

16

Viewing Test Results

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 following information applies to AJAX (Click and Script), SAP (Click and Script), Web (HTTP/HTML), Web (Click and Script), and Web Services Vuser scripts.

Note: To enable all the VuGen Web report features, it is recommended that you work with Microsoft Internet Explorer 5.0 or later.

About Viewing Test Results

When you debug a Vuser script using VuGen, you specify whether or not to generate a **Results Summary** report during script execution. The Results Summary report contains details of all the Web pages that the Vuser visited as well as any checks that the Vuser performed. Examining this information is useful when debugging the Web Vuser script. For details on running Vuser scripts using VuGen, see "Running Vuser Scripts in Stand-Alone Mode" in *Volume I-Using Service Test*.

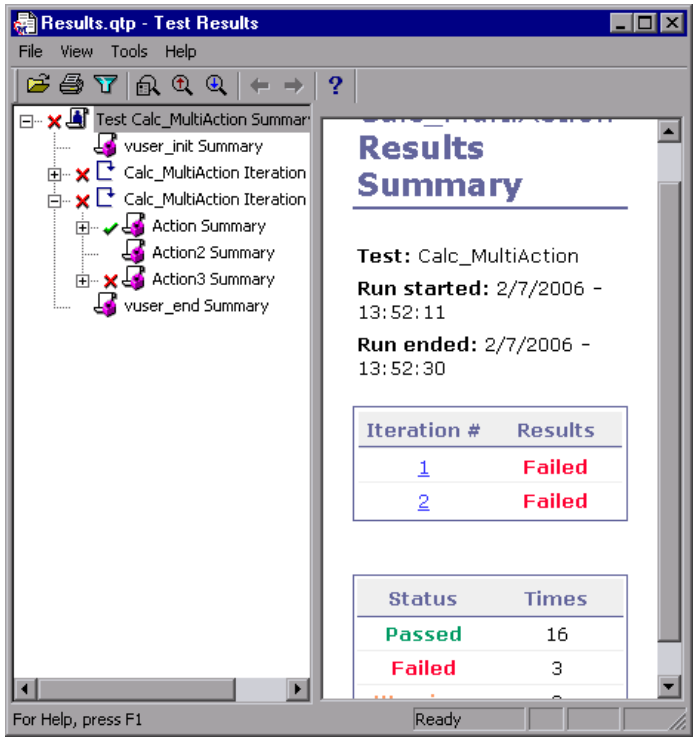
After you run a Vuser script using VuGen, you view the Results Summary report.

VuGen generates the results in VuGen report format—with a **.qtp** extension—and you view the results in the Virtual User Generator Report window. This is the recommended option because VuGen's Report window provides you with a more sophisticated interface and additional features.

You set the Display options (**Tools** > **General Options**) to specify whether or not VuGen should generate a Results Summary report, and if so, whether the report opens automatically after script execution. For details on setting the Display options, see "Running Vuser Scripts in Stand-Alone Mode" in *Volume I-Using Service Test*.

Understanding the Results Summary Report

After running your Vuser script, you view the Results Summary report. The report displays a summary of the results of the script execution.



- The left pane displays the report tree—a graphical representation of the results. In the report tree, a green check mark represents a successful step, and a red X represents a failed step.
- The right pane displays the report details—an overall summary of the script run, as well as additional information for a selected branch of the report tree.

You select a branch of the report tree to view the information for that branch.

Select this branch…	To view the following details:
Test Name Test mercury	the overall results summary of the script execution
Test Iteration Test Iteration	the execution summary for a specific iteration
Test Step or Check Link: Contact Us Text Check:	the Web page for the selected step or check in the Vuser script

You can collapse or expand a branch in the report tree in order to change the level of detail that the tree displays.

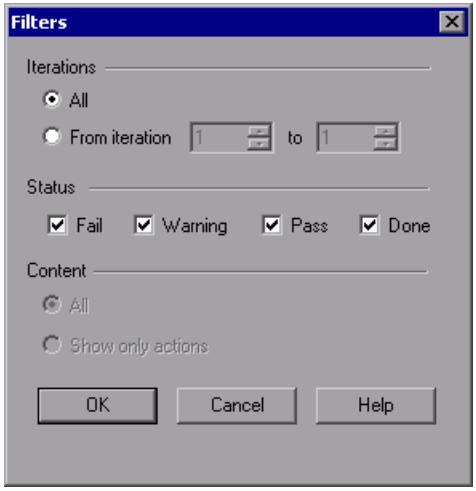
- To collapse a branch, click the Collapse (-) sign to the left of the branch you want to collapse. The report tree hides the details of the branch, and the Collapse sign changes to an Expand (+) sign.
- To collapse all the branches in the report tree, select **View** > **Collapse All**.
- To expand a branch, click the Expand (+) sign to the left of the branch you want to expand. The report tree displays the details of the branch, and the Expand sign changes to a Collapse (-) sign.
- To expand all the branches in the report tree, select **View** > **Expand All**.

Filtering Report Information

You can filter the information that is displayed in a VuGen Results Summary report. The filter can be based either on the iteration number or on the status of the iteration.

To filter the information contained in your report:

1 Click the **Filter** button on the **Report** toolbar, or select **View** > **Filters**. The Filters dialog box opens.



2 Indicate the iterations to which you want to apply the filter. The default filter option is **All**.

To limit the report to a specified range of iterations, select **From Iteration** and specify a range.

3 Specify one or more **Status** filters for the report: **Fail**, **Warning**, **Pass**, and **Done**. The Test Results window will only show the selected items.

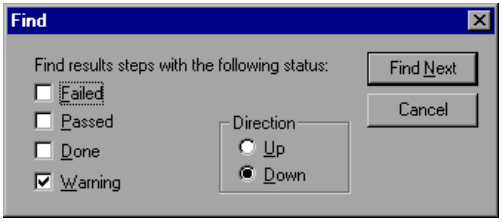
4 Click **OK** to accept the settings and filter the results.

Searching Your Results

You can search for result steps within your Test Results, by their final status: **Failed**, **Passed**, **Done**, or **Warning**. You can select more than one status for your search.

To search for a step with a specific status:

1 Select **Tools** > **Find**, or click the **Find** button on the **Report** toolbar. The Find dialog box opens.



2 Select the status (one or more) of the step that you want to find.

3 Select a search direction, **Up** or **Down**.

4 Click **Find Next**. The cursor jumps to the first match.

5 To repeat the search, click the **Find Next** button.

Managing Execution Results

You use the commands in the File menu to open, print, and save Results Summary reports.

Opening a Results Summary Report

When you run a Web Vuser script, VuGen saves the Results Summary report files in a results subfolder of the script folder. The report file has the format: **script_name.qtp**.

To open a Results Summary report:

1 Select **File** > **Open**, or click the **Open** button on the **Report** toolbar. The Open dialog box opens.

2 Select the name of the report file that you want to open, and click **Open**.

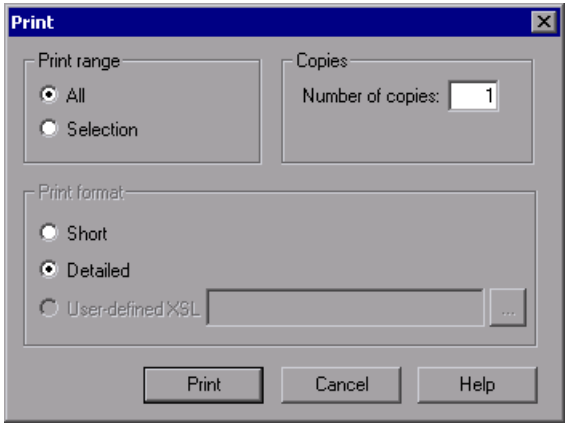
3 To open a recently viewed report, select it from the report history list on the **File** menu.

Printing Report Results

You can print a Test Results Summary report.

To print a Test Summary report:

1 Select **File** > **Print**, or click the **Print** button on the **Report** toolbar. The Print dialog box opens.



2 Select a range from the **Print Range** box:

- **All.** Prints all of the pages in the report. This includes a page for each step of an iteration.
- **Selection.** Prints the selected branch in the **Report** tree.

3 Indicate the number of copies to print.

4 Specify a **Print format**:

- **Short.** Prints a brief summary of the test results.
- **Detailed.** Prints the entire report.

5 Click **Print**.

Closing a Test Summary Report

To close a Test Summary report, select **File** > **Exit**. The Test Results window closes.

17

Managing Scripts Using Quality Center

Service Test's integration with Quality Center lets you manage Vuser scripts using Quality Center.

About Managing Scripts Using Quality Center

Service Test works together with Quality Center, HP's Web-based test management tool. HP Quality Center provides an efficient method for storing and retrieving Vuser scripts, scenarios, and results. You store scripts in a Quality Center project and organize them into unique groups.

In order for Service Test to access a Quality Center project, you must connect it to the Web server on which Quality Center is installed. You can connect to either a local or remote Web server.

For more information on working with Quality Center, refer to the *Quality Center User's Guide*.

Connecting to and Disconnecting from Quality Center

To store and retrieve scripts from Quality Center, you need to connect to a Quality Center project. You can connect or disconnect from a Quality Center project at any time during the testing process.

Connecting to Quality Center

The connection process has two stages. First, you connect to a local or remote Quality Center Web server. This server handles the connections between Service Test and the Quality Center project.

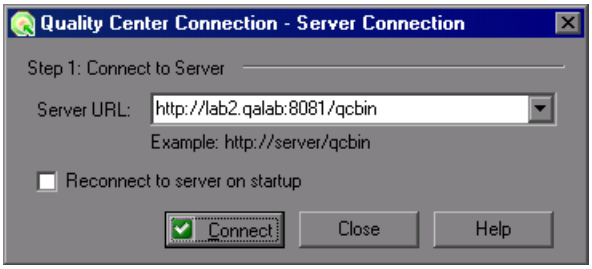
Next, you choose the project you want to access. The project stores the scripts that you created in Service Test.

Note: Quality Center projects are password protected, so you must provide a user name and a password.

To connect to Quality Center:

1 Select **Tools** > **Quality Center Connection**.

2 The Quality Center Connection - Server Connection dialog box opens.



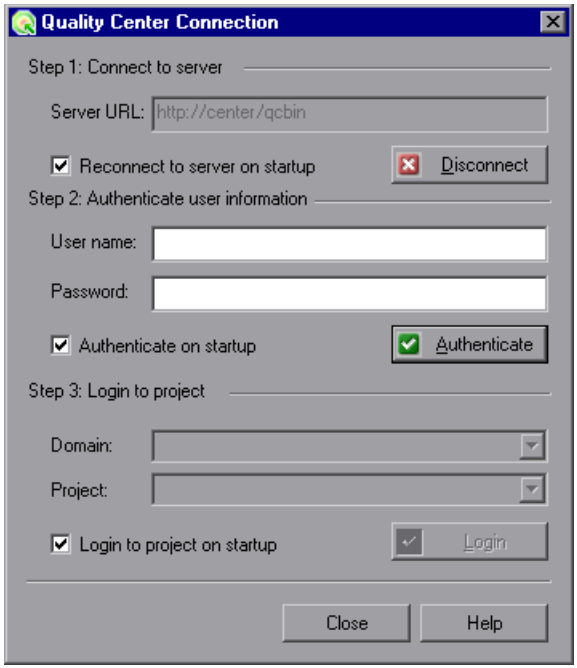
3 In the **Server URL** box, type the URL address of the Web server where Quality Center is installed.

Note: You can choose a Web server accessible via a Local Area Network (LAN) or a Wide Area Network (WAN).

---

4 To automatically reconnect to the Quality Center server the next time you open Service Test, select the **Reconnect to server on startup** check box.

5 Click **Connect**. The Quality Center Connection dialog box opens.



After the connection to the server is established, the server's name is displayed in read-only format in the Server box.

6 Authenticate your user information:

a In the **User name** box, type your Quality Center user name.

b In the **Password** box, type your Quality Center password.

c Click **Authenticate** to authenticate your user information against the Quality Center server.

After your user information has been authenticated, the fields in the Authenticate user information area are displayed in read-only format. The **Authenticate** button changes to a **Change User** button.

You can log in to the same Quality Center server using a different user name by clicking **Change User**, entering a new user name and password, and then clicking **Authenticate** again.

7 If you selected **Reconnect to server on startup** above, the **Authenticate on startup** option is enabled. To authenticate your user information automatically, the next time you open Service Test, select **Authenticate on startup**.

8 Enter the details for logging in to the project:

a In the **Domain** box, select the domain that contains the Quality Center project. Only those domains containing projects to which you have permission to connect to are displayed. (If you are working with a project in versions of TestDirector earlier than version 7.5, the **Domain** box is not relevant. Proceed to the next step.)

b In the **Project** box, enter the Quality Center project name or select a project from the list. Only those projects that you have permission to connect to are displayed.

c Click **Login**.

9 To log in to the selected project on startup, select **Login to project on startup**. This option is only enabled when the **Authenticate on startup** check box is selected.

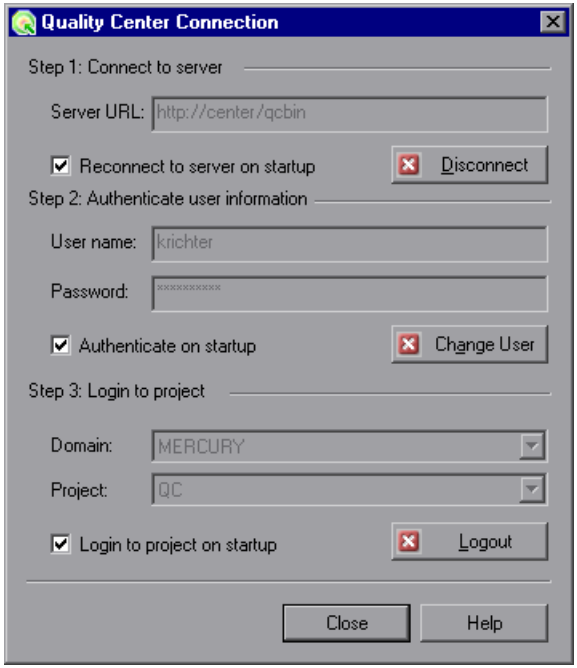
10 Click **Close** to close the Quality Center Connection dialog box.

Disconnecting from Quality Center

You can disconnect Service Test from a selected Quality Center project and Web server.

To disconnect from Quality Center:

1 Select **Tools** > **Quality Center Connection**. The Quality Center Connection dialog box opens.



2 To disconnect from the selected project, under **Login to project**, click **Logout**. If you want to open a different project while using the same server, enter the Quality Center project name in the **Project** box or select a project from the list.

3 To disconnect from the Quality Center server, under **Connect to server**, click **Disconnect**.

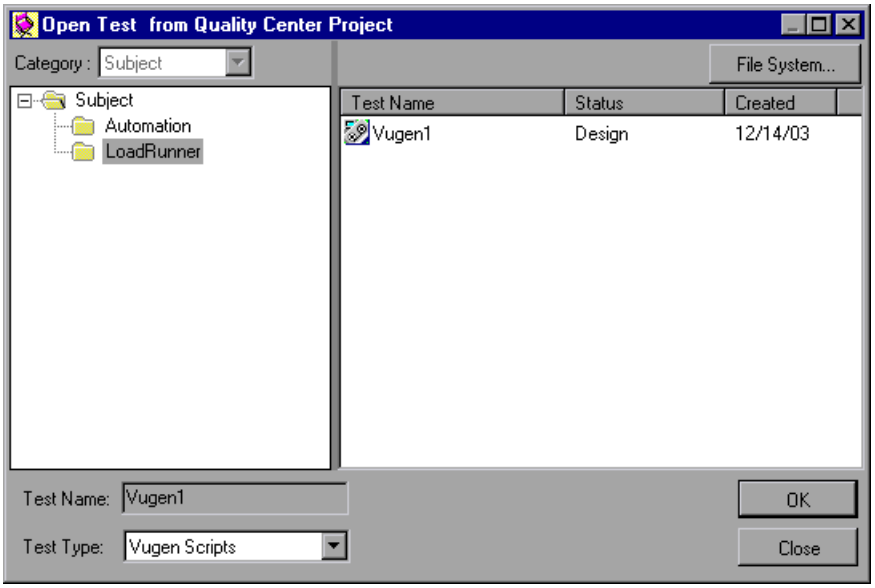
4 Click **Close** to close the Quality Center Connection dialog box.

Opening Scripts from a Quality Center Project

When connected to a Quality Center project, you can open your scripts from Quality Center. You locate tests according to their position in the test plan tree, rather than a location in the file system.

To open a script from a Quality Center project:

1. Connect to the Quality Center server. (**Tools** > **Quality Center Connection**)
2. Choose **File** > **Open**. The Open Test from Quality Center Project dialog box opens and displays the test plan tree.



Note: To open a script directly from the file system, click the **File System** button. (To return to the Open from Quality Center Project dialog box, click the **Quality Center** button.)

---

3. Click the relevant subject in the test plan tree. To expand the tree and view sublevels, double-click closed folders. To collapse the tree, double-click open folders.

   Note that when you select a subject, the scripts that belong to the subject appear in the Test Name list.

4. Select a script from the Test Name list. The script appears in the read-only Test Name box.

5. Click **OK** to open the script. Service Test loads the script and displays its name in the title bar.

Note: You can also open scripts from the recent file list in the File menu. If you select a script located in a Quality Center project, but you are not connected to that project, the Quality Center Connection dialog box opens.

Opening Tests from the Recent Files List

You can open scripts from the recent files list in the File menu. If you select a file located in a Quality Center project, but you are not connected to Quality Center or to the correct project for that file, Service Test prompts you to connect to Quality Center.

Log in to the project to continue working with the script.

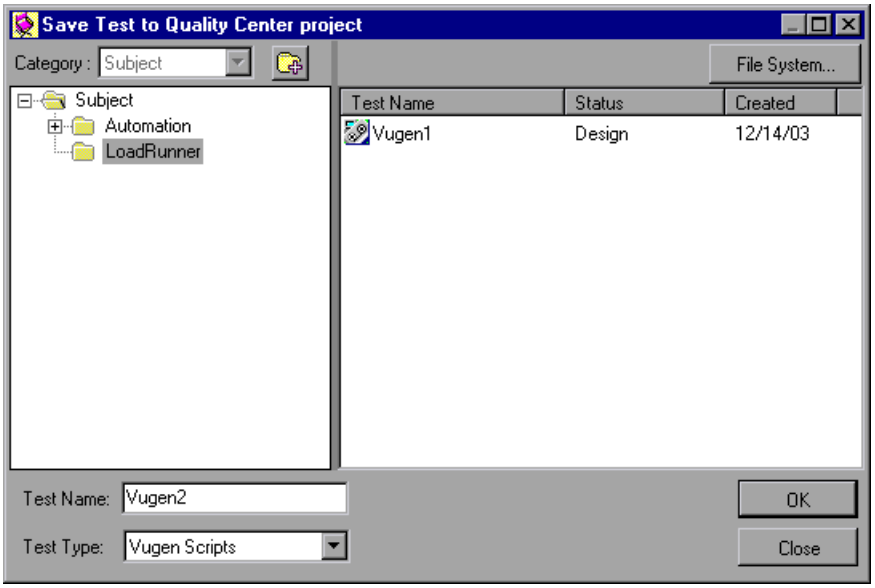
The Connect to Quality Center Project dialog box also opens if you choose to open a script that was last edited on your computer using a different user name. You can either log in using the displayed name or you can click **Cancel** to stay logged in with your current user name.

Saving Scripts to a Quality Center Project

When Service Test is connected to a Quality Center project, you can create new scripts in Service Test and save them directly to your project. To save a script, you give it a descriptive name and associate it with the relevant subject in the test plan tree. This helps you to keep track of the scripts created for each subject and lets you view the progress of test planning and creation.

To save a script to a Quality Center project:

1. Connect to the Quality Center server. (**Tools** > **Quality Center Connection**)
2. Choose **File** > **Save As**. The Save Test to Quality Center Project dialog box opens and displays the test plan tree.



To save a script directly in the file system, click the **File System** button. The Save Test dialog box opens. (From the Save Test dialog box, you may return to the Save Test to Quality Center Project dialog box by clicking the **Quality Center** button.)

3. Select the relevant subject in the test plan tree. To expand the tree and view a sublevel, double-click a closed folder. To collapse a sublevel, double-click an open folder.
4. In the Test Name box, enter a name for the script. Use a descriptive name that will allow you identify the script easily.
5. Click **OK** to save the script and close the dialog box.

   The next time you start Quality Center, the new script will appear in Quality Center's test plan tree.

Managing Script Versions

When connected to a Quality Center project with version control support, you can update and revise your automated scripts while maintaining old versions. This helps you keep track of the changes made to each script, see what was modified from one version of a script to another, or return to a previous version of the script.

You add a script to the version control data base by saving it in a project with version control support. You manage versions by checking scripts in and out of the version control database.

The script with the latest version is the script that is located in the Quality Center repository and is used by Quality Center for all test runs.

Note: The **Quality Center Version Control** options in the **File** menu are available only when you are connected to a Quality Center project database with version control support.

Adding Scripts to the Version Control Database

When you use **Save As** to save a new script in a Quality Center project with version control support, Service Test automatically saves the script in the project, checks the script into the version control database with version number 1.1.1 and then checks it out so that you can continue working.

The status bar indicates each of these operations as they occur. Note, however, that saving your changes to an existing script does not check them in. Even if you save and close the script, it remains checked out until you choose to check it in. For more information, see "Checking Scripts Out of the Version Control Database" on page 284.

Checking Scripts Out of the Version Control Database

When you choose **File** > **Open** to open a script that is currently checked in to the version control database, it is opened in read-only mode.

Note: The Open Test from Quality Center Project dialog box displays icons that indicate the version control status of each script in your project. For more information, see "Opening Scripts from a Quality Center Project" on page 280.

You can review the checked-in script. You can also run the script and view the results.

To modify the script, you must check it out. When you check out a script, Quality Center copies the script to your unique check-out directory (automatically created the first time you check out a script), and locks the script in the project database. This prevents other users of the Quality Center project from overwriting any changes you make to the script. However, other users can still run the version that was last checked in to the database.

You can save and close the script, but it remains locked until you return the script to the Quality Center database. You can release the script by either checking the script in, or undoing the check out operation. For more information on checking scripts in, see "Checking Scripts into the Version Control Database" on page 285. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 291.

By default, the check out option accesses the latest version of the script. You can also check out older versions of the script. For more information, see "Using the Version History Dialog Box" on page 287.

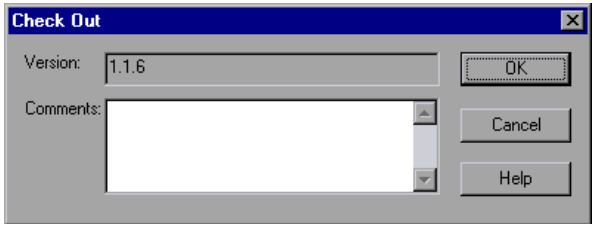
To check out the latest version of a script:

1 Open the script you want to check out. For more information, see "Opening Scripts from a Quality Center Project" on page 280.

Note: Make sure the script you open is currently checked in. If you open a script that is checked out to you, the **Check Out** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

---

2 Choose **File** > **Quality Center Version Control** > **Check Out**. The Check Out dialog box opens and displays the script version to be checked out.



3 You can enter a description of the changes you plan to make in the **Comments** box.

4 Click **OK**. The read-only script closes and automatically reopens as a writable script.

Checking Scripts into the Version Control Database

While a script is checked out, Quality Center users can run the previously checked-in version of your script. For example, suppose you check out version 1.2.3 of a script and make a number of changes to it and save the script. Until you check the script back in to the version control database as version 1.2.4 (or another number that you assign), Quality Center users can continue to run version 1.2.3.

When you have finished making changes to a script and you are ready for Quality Center users to use your new version, you check it in to the version control database.

Note: If you do not want to check your changes into the Quality Center database, you can undo the check-out operation. For more information, see "Canceling a Check-Out Operation" on page 291.

When you check a script back into the version control database, Quality Center deletes the script copy from your checkout directory and unlocks the script in the database so that the script version will be available to other users of the Quality Center project.

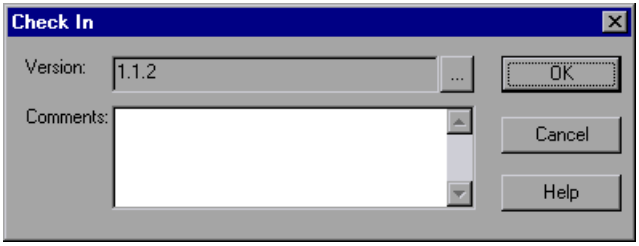
To check in the currently open script:

1 Confirm that the currently open script is checked out to you. For more information, see "Viewing Version Information For a Script" on page 287.

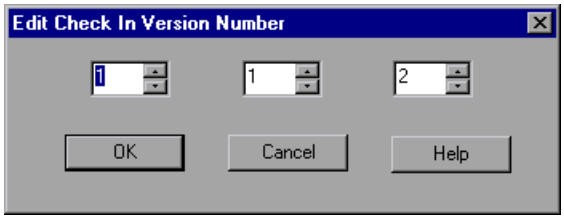
Note: If the open script is currently checked in, the **Check In** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

---

2 Choose **File** > **Quality Center Version Control** > **Check In**. The Check In dialog box opens.



3 Accept the default new version number and proceed to step 7, or click the browse button to specify a custom version number. If you click the browse button, The Edit Check In Version Number dialog box opens.



4 Modify the version number manually or using the up and down arrows next to each element of the version number. You can enter numbers 1-900 in the first element. You can enter numbers 1-999 in the second and third elements. You cannot enter a version number lower than the most recent version of this script in the version control database.

5 Click **OK** to save the version number and close the Edit Check In Version Number dialog box.

6 If you entered a description of your change when you checked out the script, the description is displayed in the **Comments** box. You can enter or modify the comments in the box.

7 Click **OK** to check in the script. The script closes and automatically reopens as a read-only file.

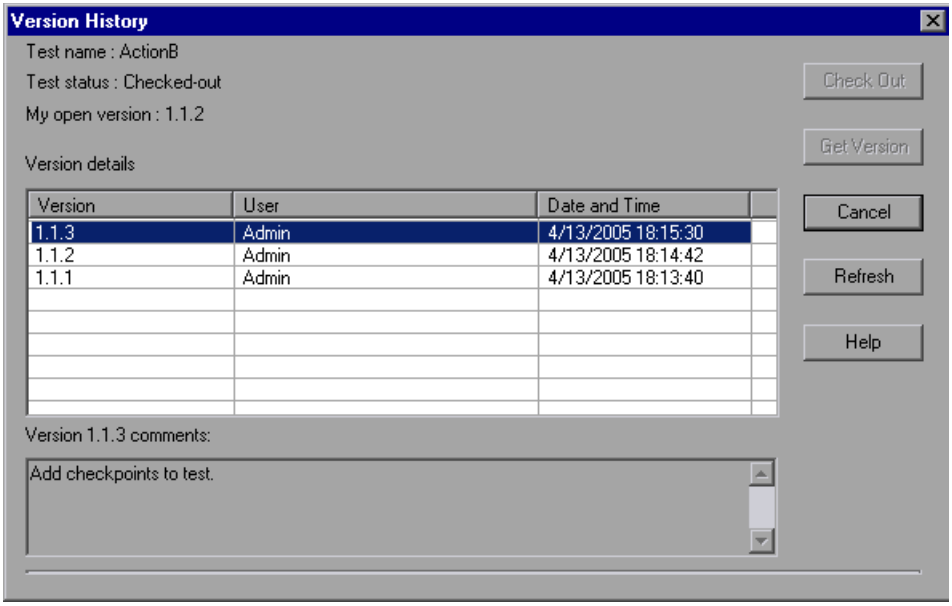
Using the Version History Dialog Box

You can use the Version History dialog box to view version information about the currently open script and to view or retrieve an older version of the script.

Viewing Version Information For a Script

You can view version information for any open script that has been stored in the Quality Center version control database, regardless of its current status.

To open the Version History dialog box for a script, open the script and choose **File** > **Quality Center Version Control** > **Version History**.



The Version History dialog box provides the following information:

- **Test name**. The name of the currently open script.
- **Test status**. The status of the script. The script can be:
  - **Checked-in**. The script is currently checked in to the version control database. It is currently open in read-only format. You can check out the script to edit it.
  - **Checked-out**. The script is checked out by you. It is currently open in read-write format.
  - **Checked-out by <another user>**. The script is currently checked out by another user. It is currently open in read-only format. You cannot check out or edit the script until the specified user checks in the script.
- **My open version**. The script version that is currently open on your QuickTest computer.

- **Version details.** The version details for the script.
  - **Version**. A list of all versions of the script.
  - **User**. The user who checked in each listed version.
  - **Date and Time**. The date and time that each version was checked in.
- **Version comments**. The comments that were entered when the selected version was checked in.

Working with Previous Script Versions

You can view an old version of a script in read-only mode or you can check out an old version and then check it in as the latest version of the script.

To view an old version of a script:

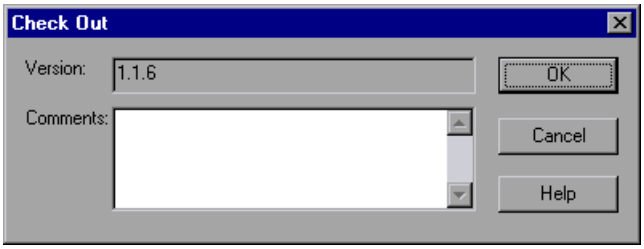
1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 280.
2. Choose **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version you want to view in the **Version details** list.
4. Click the **Get Version** button. QuickTest reminds you that the script will open in read-only mode because it is not checked out.
5. Click **OK** to close the QuickTest message. The selected version opens in read-only mode.

Tips: To confirm the version number that you now have open in QuickTest, look at the **My open version** value in the Version History dialog box.

After using the **Get Version** option to open an old version in read-only mode, you can check-out the open script by choosing **File** > **Quality Center Version Control** > **Check Out**. This is equivalent to using the **Check Out** button in the Version History dialog box.

To check out an old version of a script:

1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 280.
2. Choose **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version you want to view in the **Version details** list.
4. Click the **Check Out** button. A confirmation message opens.
5. Confirm that you want to check out an older version of the script. The Check Out dialog box opens and displays the version to be checked out.



6. You can enter a description of the changes you plan to make in the **Comments** box.
7. Click **OK**. The open script closes and the selected version opens as a writable script.
8. View or edit the script as necessary.
9. If you want to check in your script as the new, latest version in the Quality Center database, choose **File** > **Quality Center Version Control** > **Check In**. If you do not want to upload the modified script to Quality Center, choose **File** > **Quality Center Version Control** > **Undo Check out**.

   For more information on checking in scripts, see "Checking Scripts into the Version Control Database" on page 285. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 291.

Canceling a Check-Out Operation

If you check out a script and then decide that you do not want to upload the modified script to Quality Center you should cancel the check-out operation so that the script will be available for check out by other Quality Center users.

To cancel a check-out operation:

1. If it is not already open, open the checked-out script.
2. Choose **File** > **Quality Center Version Control** > **Undo Check out**.
3. Click **Yes** to confirm the cancellation of your check-out operation. The check-out operation is cancelled. The checked-out script closes and the previously checked-in version reopens in read-only mode.

18

Managing Scripts with HP Performance Center

VuGen's integration with HP Performance Center lets you upload and download scripts to and from the Performance Center server.

The following information applies to all Vuser Scripts.

About Managing Scripts with HP Performance Center

VuGen provides integration with Performance Center, HP's Web-enabled global load testing tool that allows you to test your system from different geographical locations.

You can upload and download scripts to and from Performance Center using VuGen's user interface. You upload scripts to Performance Center in order to add them to your Vuser Scripts list and use them in your test. You can also download scripts to edit them or save them locally.

In order for VuGen to access Performance Center for either uploading or downloading, you must connect to the server upon which Performance Center is installed.

For further information about working with Performance Center, refer to the *Performance Center User's Guide*.

Connecting VuGen to Performance Center

VuGen works together with Performance Center to provide an efficient method for uploading and downloading Vuser scripts to and from Performance Center. In order for VuGen to access a Performance Center project, you must first connect it to the Web server on which Performance Center is installed. You can then upload or download Vuser scripts. For more information, see:

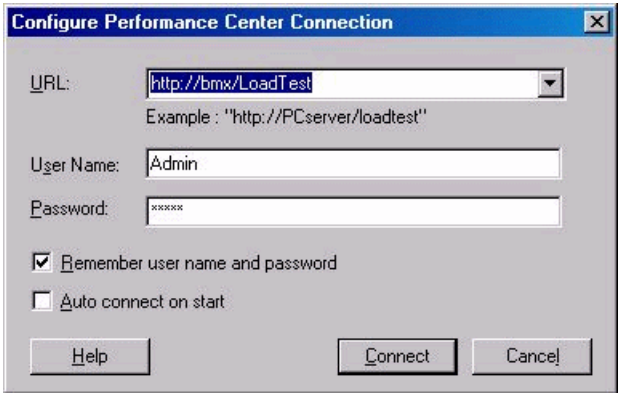
➤ "Uploading Vuser Scripts" on page 296.

➤ "Downloading Vuser Scripts" on page 301.

You connect to Performance Center using the Configure Performance Center Connection dialog box.

To connect VuGen to Performance Center:

1 In VuGen, select **Tools** > **Configure Performance Center Connection**. The Configure Performance Center Connection dialog box opens.



2 In the URL box, type the URL address of the Web server on which Performance Center is installed. The URL address should be in the format: http://<server_name>/loadtest

3. Enter your user name and password. Contact your Performance Center administrator if you need assistance.
4. To automate the login process, select **Remember User Name and Password**. The specified username and password are saved to the registry, and displayed each time you open the dialog box.
5. To automatically open the connection to the Performance Center server when you start VuGen, select **Auto connect when VuGen starts**. VuGen attempts to connect to Performance Center using the configuration information displayed.
6. Click **OK** to connect to Performance Center. The Performance Center Connection dialog box displays the connection status.

   Once the connection is established, all the fields are displayed in read-only format.

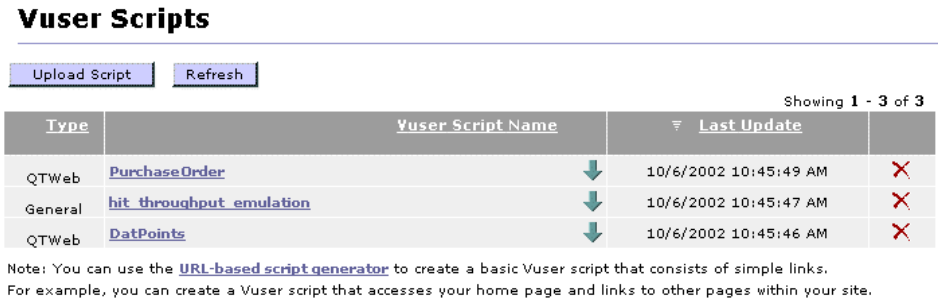
Note: If the connection fails, a dialog box displays the reason for the connection failure.

You cannot be connected to Performance Center and Quality Center at the same time.

Uploading Vuser Scripts

In order to use Vuser scripts in your Performance Center project, you need to upload the Vuser scripts to the Performance Center server. The Vuser Scripts list on the Performance Center server displays all the stored Vuser scripts that are available for your project.

To open the Vuser Scripts page, select **Vuser Scripts** from the **Projects** menu.



If you have already added Vuser scripts to the Vuser Scripts list, Performance Center displays them on the Vuser Scripts page. This list represents all the Vuser scripts that are available for use by Vusers during a load test. Scripts uploaded using VuGen are automatically added to the Vuser Scripts list.

If you do not have Vuser scripts in the Vuser Scripts list and you want to add a new script, you can add scripts by:

- Uploading a Vuser Script from VuGen
- Uploading a Vuser Script from Performance Center

Verifying your Version of VuGen

In order to process the upload, your version of VuGen must be properly configured, and you must connect VuGen to Performance Center.

To check for proper upload configuration in VuGen:

1. Start VuGen and open a new or existing VuGen script.
2. Select **Tools**.

If the menu item **Configure Performance Center Connection** is available on the drop-down menu, your version of VuGen is enabled to upload scripts.

If your version of VuGen is not enabled to do uploads or if you do not have VuGen installed on your machine, you need to install a newer version of VuGen. It is recommended that you uninstall your older version. To uninstall VuGen, choose the uninstall option under the Virtual User Generator from the **Start** menu.

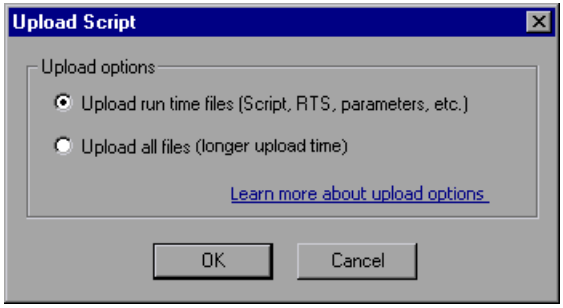
To install a newer version of VuGen:

1. On the Performance Center User site, select **Downloads** from the **Miscellaneous** menu.
2. Select **Standalone LoadRunner VuGen**.
3. Follow the download instructions.

Once you have a version of VuGen that is enabled, you can upload existing scripts or record new scripts to upload.

Upload Options

You can upload Vuser scripts from within VuGen to the Performance Center Web site script repository. Depending on your requirements, you can do a partial or a complete upload. The upload options are:



➤ **Upload run time files.** First VuGen deletes all main script files (*usr*, *c*, *cfg*, and *xml* files) from the server. It does not delete data files or old recorded data. Next, VuGen uploads the script files, the run-time settings, and the parameter files.

➤ **Upload all files.** First VuGen deletes all script and data files from the server. VuGen then uploads the current script and data files, including the recording data and the replay result directories.

Uploading the run time files only is quicker since VuGen only uploads the script files—not all of the recording data and the replay results.

Note: If you previously downloaded script files, VuGen by default uploads only the files that were downloaded. If you want to upload newly created files, for example, you replayed the downloaded script to create snapshots, you must specify that all files are uploaded.

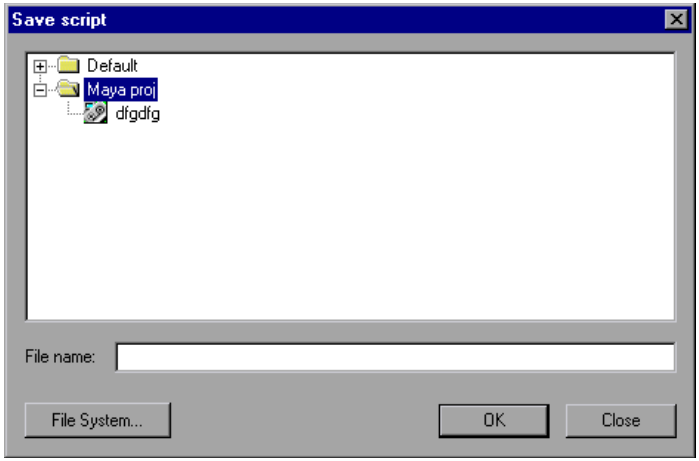
Uploading a Vuser Script from VuGen

Once you are connected to VuGen, you can upload your script files to the Performance Center server. For more information, see "Connecting VuGen to Performance Center" on page 294.

If VuGen is not connected to Performance Center, you can save the Vuser script files locally to the file system. Later, when VuGen is connected to Performance Center, open the script in VuGen and upload it to Performance Center as described below.

To upload a Vuser script to Performance Center from VuGen:

1 Choose **File** > **Save** in VuGen. The Save script dialog box opens.



2 Select the project where you want to save the script. Type name for the script in the **File name** box.

Note: File names can only consist of English letters, digits, or the underscore character, and cannot exceed 250 characters.

---

3 Click **OK**. The Upload Script dialog box opens. Select one of the Upload Options, **Upload run time files** or **Upload all files**.

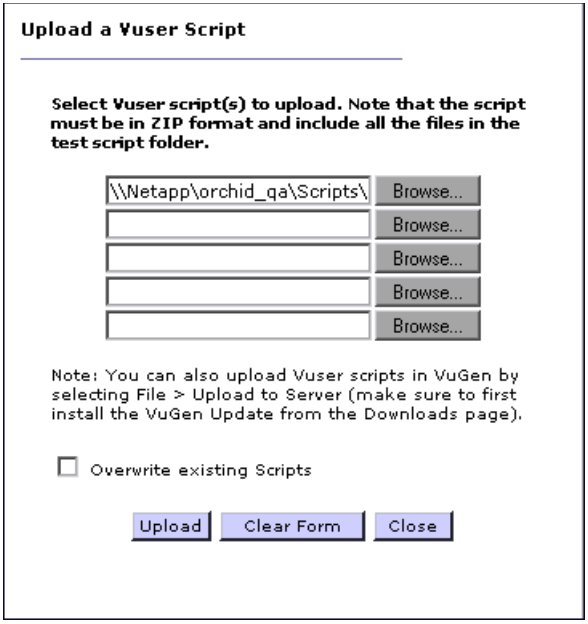
4 Click **OK** to upload the files to the Performance Center server.

Uploading a Vuser Script from Performance Center

If your installation does not include VuGen, you can still upload scripts using the Upload Script function in Performance Center's **Vuser Scripts** page.

To upload a Vuser script from Performance Center:

1. Open the Vuser Scripts page.
2. Click **Upload Script**. The Upload a Vuser Script dialog box opens.



3. Click a **Browse** button to browse to the zip file containing each Vuser script you want to upload. Note that the zip file must contain the complete contents of the Vuser script folder, including the Vuser script file (".usr" file) itself and all related data files.
4. Select the zip file, and click **Open**.
5. Check **Overwrite existing Scripts** if you want to replace a script that already exists in the Vuser Scripts list.
6. Click **Upload** to upload the script and add it to your Vuser Scripts list.

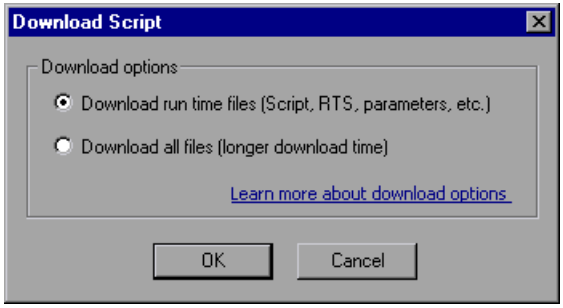
Downloading Vuser Scripts

VuGen works together with Performance Center to provide an efficient method for downloading Vuser scripts from Performance Center for editing, and automatically opening them in VuGen. You can choose to download only the script run time files, or the complete files including the recording data and replay results.

In order for VuGen to access a Performance Center project, you must first connect it to the Web server on which Performance Center is installed. You can then select the script files that you want to download. You connect to Performance Center from the Configure Performance Center Connection dialog box. For more information, see "Connecting VuGen to Performance Center" on page 294.

Download Options

You can download Vuser scripts from the Performance Center script repository to VuGen. Depending on your requirements, you can do a partial or a complete download. The download options are:



- **Download run time files.** VuGen downloads the script files only, allowing faster downloads. This includes the script file, run-time settings, and parameter files.
- **Download all files.** VuGen downloads the script and data files, including the recording data and the replay result directories.

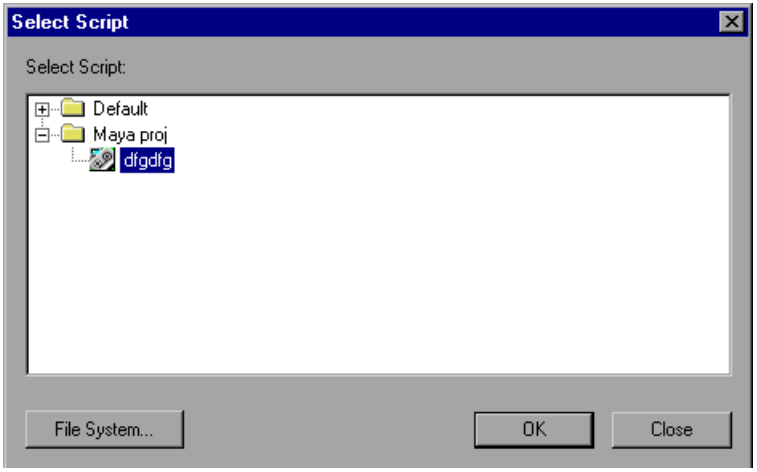
A partial download of run time files only is quicker since VuGen only downloads the script files. If you download all the script and data files, the transfer will take more time.

Downloading a Vuser Script from VuGen

Once you are connected to the Performance Center server, you can download your script files to the VuGen.

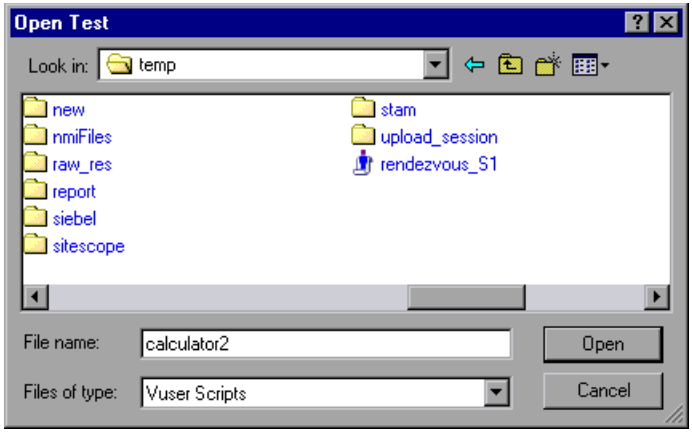
To download a Vuser script from Performance Center:

1. Connect to the Performance Center server. For more information, see "Connecting VuGen to Performance Center" on page 294.
2. In VuGen, select **File** > **Open**. The Select Script dialog box opens.



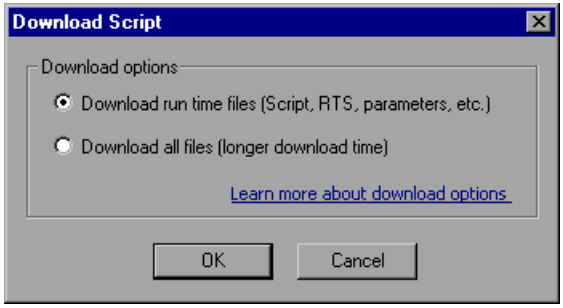
3. Select the script that you want to download.

To select a script from a local drive (even if you are connected to Performance Center), click **File System**. The Open Test dialog box opens.



Browse to the file that you want to download, and click **Open**. The Performance Center Select Script dialog box reopens. Select the script that you want to download.

4 Click **OK**. The Download Script dialog box opens.



5 Select a download option: **Download run time files** or **Download all files**.

6 Click **OK** to download the files from Performance Center. When the download is complete, the dialog box closes and VuGen displays the script.

By default, downloaded files are saved to your temp directory. To save them to a different directory, click **Save** and specify a directory.

Part III

SOA, BPT, and Web Services Testing

19

Understanding the SOA Test Types

You use HP Service Test to create tests for your Web Services.

The following information only applies to Web Services/SOA Vuser scripts.

About SOA Test Types

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 enabling the rapid development and deployment of new applications.

Using Service Test, you create test scripts for testing your SOA environment. You can use a test generation wizard to automatically generate scripts, or create the scripts manually.

To automatically generate test scripts, you use the SOA Test Generator. A wizard guides you through the process of selecting testing aspects such as interoperability with different toolkits, boundary testing, and standard compliance. For more information, see Chapter 23, “Using the SOA Test Generator.”

To manually create scripts, you begin by creating an empty script. Then you add content to the script either by recording a session, analyzing network traffic, or manually inserting calls to the Web service as described in Chapter 22, "Adding Content to Web Services Scripts."

An additional test type is BPT (Business Process Testing). BPT is a methodology in which several tasks are combined to create a complete business process. For more information, see "BPT (Business Process Testing)" on page 356.

For manual scripts, you use Service Test to create one of the following scripts.

- **Single Protocol Script.** A script that emulates SOAP traffic by sending SOAP requests to the Web service.
- **Multi Protocol Script.** A script that emulates several protocols in a single script. For example, if your environment contains a client that accesses a Web Services and Web pages, choose both the Web Services and Web (Click and Script) protocols.

Getting Started with Web Services Vuser Scripts

This section provides an overview of the process of developing a Web Services / SOA Vuser script.

To develop a test script:

1. **Create a new Web Services script.**

   Create a new script using the SOA Test Generator, or manually create a new single or multiple protocol script, or a Business Process Testing component.

2. **Add content to the script.**

   Add content to the script (excluding the SOA Test Generator). For details, see Chapter 22, "Adding Content to Web Services Scripts."

3. **Set properties, values, and checkpoints.**

   Enhance the script by customizing the step properties, inserting argument values, and setting checkpoints. For details, see Chapter 25, "Working in the Web Service Call View."

4 Parametrize your script.

Parameterization lets you replace constant values with a variable to substitute new values for each iteration. To parameterize a value, double-click on a step to open its properties and click the **ABC** icon adjacent to the value box. For complex type elements, use the XML parameter type as described in "Setting Properties for XML Parameters" on page 180.

5 Validate the service.

Test your service for compliancy. For more information, see "Performing WS-I Validation" on page 333.

6 Configure the Run-Time settings.

The Run-Time settings control the script's behavior during execution. These settings include Web Service-specific settings (client emulation) and General settings—run logic, pacing, logging, and think time.

For information about the Web Service-specific settings, see "Setting Web Service Toolkit Run-Time Settings" on page 450, and Chapter 13, "Configuring Run-Time Settings."

7 Verify that the script is functional.

Replay the script in Service Test to verify that it runs correctly.

For details about replaying the script, see Chapter 15, "Running Vuser Scripts in Stand-Alone Mode."

8 Save the script.

Save the script in the file system or in a Quality Center repository. If you save the scripts in Quality Center, you can associate them to a test set and perform functional and regression testing directly from Quality Center. For more information about Quality Center and its integration with scripts, see "Working with Service Test Management" on page 361.

After you prepare a script, you are ready to use it for your testing. For more information, see "Using Your Script" on page 355.

Use Quality Center to manage all of your tests while tracking defects and requirements. For more information, see www.hp.com or contact your sales representative.

20

Working with Web Services and SOA Tests

After creating a Web Services script, you can view it in either Script view or Tree view. Within these views, you can modify the script and its properties.

The following information only applies to Web Services Vuser scripts.

About Working with Web Services and SOA Tests

After you create a script, you can view its contents in either Tree view or Script view. The tree view is a graphical view, while Script view displays the actual functions in the script.

When viewing your script, you can examine it and determine if it needs to be enhanced in any way. The most common enhancements are transactions and parameterization.

Transactions let you mark a group of actions to be measured to check the applications's performance. For example, if you want to check the time it took for a service to update an address, you mark those actions as a transaction. For more information, see "Inserting Transactions into a Vuser Script" on page 126.

Parameterization is the replacing of constants with parameters. This is useful for testing your service with different values, or passing information from one step to another. For more information, see "Parameterizing Scripts" on page 315.

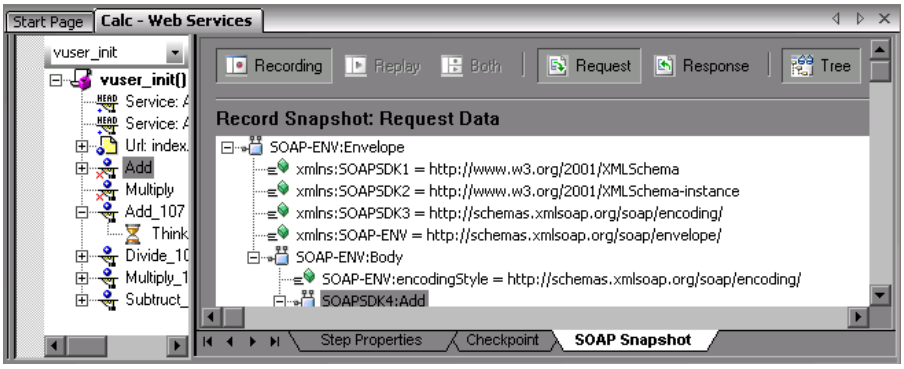
Viewing and Editing Scripts

You can view and edit all of the scripts that you created both manually and automatically in the Service Test window.

You can view a script in either Tree View or Script View. Tree view displays the steps of the script in a graphical interface, while the Script view shows all steps, including the actual **web_service_call** functions that emulate your service. Script view is ideal for advanced users that require more flexibility within the script.

Tree View

The Tree view shows a graphical representation of each one of the script's steps.



When you select a step, Service Test displays information about the step in several tabs:

➤ **Step Properties.** The properties and argument values of the Web service call. This tab allows you to modify the properties of an existing step. See "Understanding Web Service Call Properties" on page 385.

- **Checkpoint.** A list of checkpoints defined for the step. See "Setting Checkpoints" on page 409.
- **SOAP Snapshot.** A snapshot of the SOAP request and response for both record and replay. See "Viewing Web Services SOAP Snapshots" on page 382.

For more information about these tabs, see Chapter 25, "Working in the Web Service Call View."

To view a script in Tree view:

1. Click the **Tree** button or choose **View** > **Tree View**.
2. In the upper left box, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new action choose **Actions** > **Create New Action**.
3. In the left pane, select the step or sub-node that you want to view or modify.
4. Select the **Step Properties** tab in the right pane to view or modify the properties.
5. Select the **Snapshot** tab to view the step's SOAP header and body. To display a specific replay iteration, choose **View** > **Snapshot** > **Select Iteration**.
6. To add additional Web Service steps, click the **Add Service Call** button. For more information, see "Adding New Web Service Calls" on page 350.
7. To insert advanced functionality, such as JMS queue retrieval and SAML security, choose **Insert** > **Add Step** and choose the appropriate step. For more information, see Chapter 26, "Setting Advanced Properties for Web Service Scripts."
8. To replace argument values with parameters, go to the **Step Properties** tab. Select the node whose value you want to replace in the script, and click the **ABC** icon to the right of the **Value** box.
9. To set a checkpoint, click the **Checkpoint** tab. For more Information, see "Setting Checkpoints" on page 409.

Script View

The Script view shows the actual functions that were generated in the script. You can expand or collapse each of the **web_service_call** functions to view only the functions that interest you.

To view a script in Script view:

1. Click the **Script** button or choose **View** > **Script View**.
2. In the left pane, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new section choose **Actions** > **Create New Action**.
3. To add additional Web Service steps at the location of the cursor, click the **Add Service Call** button. For more information, see Chapter 25, “Working in the Web Service Call View.”
4. To insert advanced functionality, such as JMS queue retrieval and SAML security, choose **Insert** > **Add Step** and choose the appropriate step. For more information, see Chapter 26, “Setting Advanced Properties for Web Service Scripts.”
5. To replace argument values with parameters, select the value you want to replace in the script, and select **Replace with Parameter** from the right-click menu.

For more information about the functions, refer to the *Online Function Reference* (**Help** > **Function Reference**) or select a function and click F1.

Parameterizing Scripts

Service Test supports parameterization for all of the argument values. Parametrization lets you substitute the original values with external values. This is useful for testing your service with different values, or passing information from one step to another. For an overview on parameterization, see Chapter 9, "Working with VuGen Parameters."

If your arguments are the simple, non-array type, you can replace them with a simple parameter. For example, if you want to test a service that does addition, you can substitute each of the input arguments with a parameter, and store the values in a file or a table.

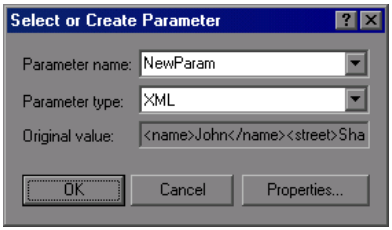
If, however, your arguments are a complex structure with many values, you can use an XML type parameter to replace the entire structure with a single parameter. You can create several value sets for the XML type parameter and assign a different value set for each iteration. For more information, see "XML Parameter Types" on page 151.

Using parameters, you can pass the output value from one operation, as input for a later operation. For more information, see "Using Web Service Output Parameters" on page 404.

To replace a constant value with a parameter:

1. Switch to the **Step Properties** tab and select the parent or child element whose value you want to parameterize.

2 Under the **Input Arguments** node, select the argument you want to parameterize. In the right pane, click the **ABC** icon in the **Value** box. The Select or Create Parameter dialog box opens.



3 Specify a parameter name and type.

4 Click **Properties** to set the type of parameter—File, XML, and so on—and to assign values.

For more information, see Chapter 9, “Working with VuGen Parameters.”

21

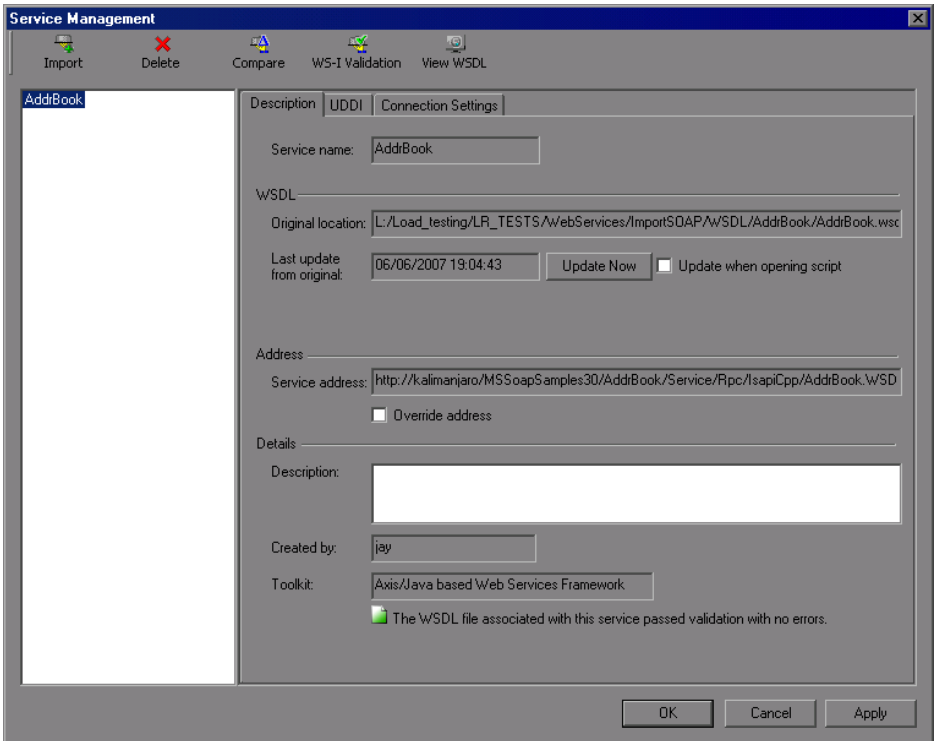
Managing Web Services

Service Test provides utilities that let you validate and manage the WSDL files associated with your service entries.

The following information only applies to Web Services and SOA Vuser scripts.

About Managing Web Services Vuser Scripts

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, Service Test 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.

To view the copy of the locally saved WSDL in Internet Explorer, click the **View WSDL** button.

Note: All validations and modifications to WSDL files are done on the working copy. If you want to replace the imported WSDL file with a newer version, use the **Update Now** option described in "Description" on page 319.

To open the Service Management window, choose **SOA Tools** > **Manage Services** or click the Manage Services toolbar button.

The Service Management window provides an interface for:

- Viewing and Setting Service Properties
- Importing Services
- Deleting Services
- Comparing WSDL Files
- Performing WS-I Validation
- Viewing WSDL Files

Viewing and Setting Service Properties

The Service Management window lets you view and modify information about the imported WSDLs. It shows details about the selected service entry in the following tabs:

- Description
- UDDI
- Connection Settings

Description

The Service Management window's **Description** tab displays information about the service: WSDL, Endpoint Address, and Details. You can add annotations or notes about your service, in the Details area.

WSDL

The WSDL area provides information about the location of the WSDL, and the date that it was last imported.

- **Original location.** The original source of the WSDL file (read-only).
- **Service name.** The name of the Web Service.
- **Last update from original.** (For services **not** imported from Quality Center) The last date that the local copy was updated with a WSDL file from the original source.
  - To manually update the working copy of the WSDL, click **Update Now**. Service Test backs up the existing WSDL and updates it from the location indicated above.
  - To instruct Service Test to update the WSDL every time you open the script, select **Update when opening script**.
  - date of the last of the service from QC
- **Last updated from QC.** (For services imported from Quality Center) The last date that the service was updated from Quality Center.

Address

- **Service address.** An endpoint address to which the request is sent.

  If you want to override the endpoint specified in the WSDL file, choose **Override address** and specify a different address in the **Service address** box.

  This is useful for implementing emulated services. Service Test uses the override address as targetAddress for the Web Service call. This overriding affects all Web Service calls. To use a different target address for a particular Web Service call, you specify it in that step's properties. For more information, see "Using Emulated Services in Vuser Scripts" on page 469.

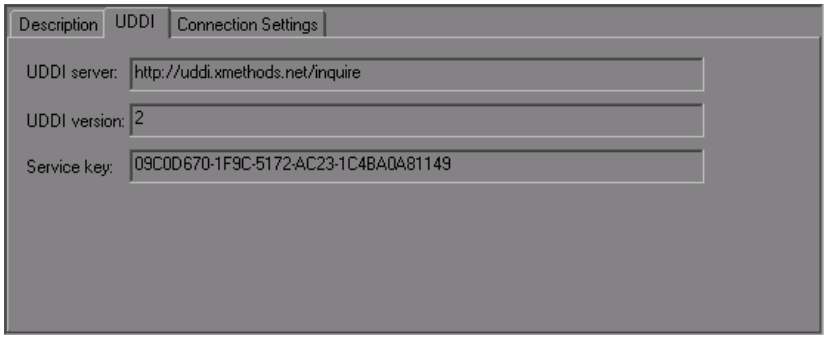
Details

- **Description.** A description of the Web service, taken by default from the WSDL file. This text area is editable.
- **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.

UDDI

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.

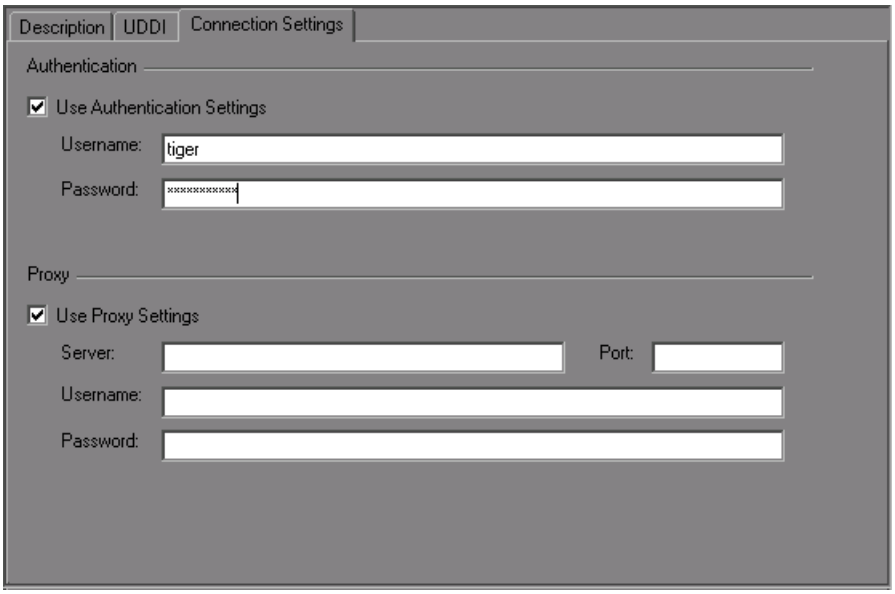
Connection Settings

In some cases WSDLs reside on secure sites requiring authentication. In certain instances, the WSDL is accessed through a proxy server.

Service Test 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 and Generic toolkits

It is recommended that you enter the authentication or proxy information while importing the WSDL. If however, the settings changed, you can modify them through the Service Manager's **Connection Settings** tab.



For more information about setting the connection information while importing the WSDL, see "Connection Settings" on page 324.

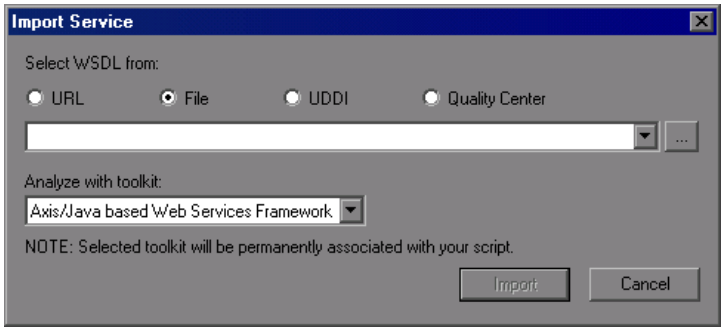
Importing Services

Service Test 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.

When importing a file, you specify the following information:

- **Source.** the source of the WSDL: URL, File, UDDI, or Quality Center
- **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 (only available for the first service added to the script)

➤ **Connection Settings.** authentication or proxy server information (optional)



If Service Test 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.

Source

When specifying a WSDL, you can indicate the source:

➤ **URL.** The complete URL of the service.

➤ **File.** The complete path and name of the WSDL file.

➤ **UDDI.** Universal Description, Discovery, and Integration—a universal repository for services. For more information, see "Specifying a Service on a UDDI Server" on page 325.

➤ **Quality Center.** A service stored in the Quality Center repository. For more information, see "Choosing a Service from Quality Center" on page 326.

Service Test supports URL or UDDI paths that are secure, requiring authentication or accessed through proxy servers. For more information, see "Specifying a Service on a UDDI Server" on page 325.

Location

In the Location box, you specify the path or URL of the WSDL.

For the URL or UDDI options, make sure to insert a complete URL—not a shortened version. Click the **Browse** button to the right of the text box to open the default browser.

For a file, click the **Browse** button to the right of the text box to locate the WSDL on the file system.

For Quality Center, click the **Quality Center Connection** button to specify a server URL and to initiate a connection. For more information, see "Choosing a Service from Quality Center" on page 326.

Toolkit

Choosing a toolkit instructs Service Test to send real client traffic using an actual toolkit—not an emulation. Once you select a toolkit, it becomes permanently associated with the script for all subsequent recordings, imports, and replays.

Service Test supports the .NET Framework with WSE 2 version SP3 and Axis/Java based Web Services Framework toolkits. For .NET and Axis/JAva toolkits, Service Test imports, records, and replays the script using the actual toolkit.

For services that do not work with the .NET or Axis/Java toolkits, Service Test provides a generic toolkit. For replay, you can configure the generic toolkit to emulate toolkits such as MS SOAP, .NET, and Axis. For more information, see "Setting Web Service Toolkit Run-Time Settings" on page 450.

Connection Settings

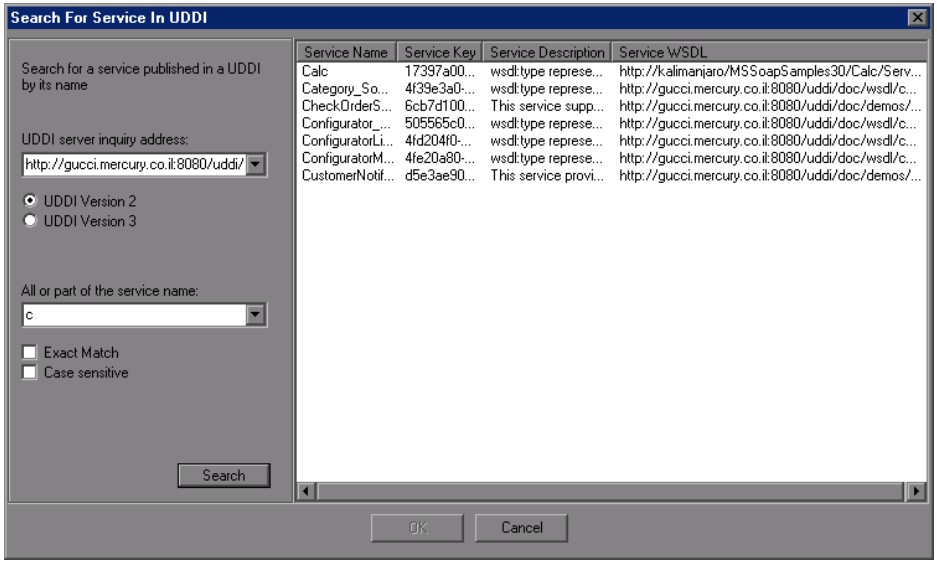
When importing WSDL files from a URL or UDDI, the WSDL may require authentication if it resides in a secure location. In certain cases, the access to the WSDL may be through a proxy server. Using the Connection Settings button, you can specify this information. For more information, see "Specifying WSDL Connection Settings" on page 327.

Specifying a Service on a UDDI Server

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.

Your Web Service client can use broker services such as the UDDI, to search for a required WSDL-based service. Once located, you bind to the server and call the service provider.

Click the **Browse** button to open the Search for Service in UDDI dialog box.



To search for a service on a UDDI:

1. In the **UDDI server inquiry address** box, enter the URL of the UDDI server.
2. Specify the UDDI version.
3. Specify the name or part of the name of the service. Select **Exact Match** or **Case Sensitive** to refine your search, if they are applicable. To perform a wildcard search, use the percent (%) character.
4. Click **Search**. Service Test displays all of the matching results.
5. Double-click on a service in the list to import it.

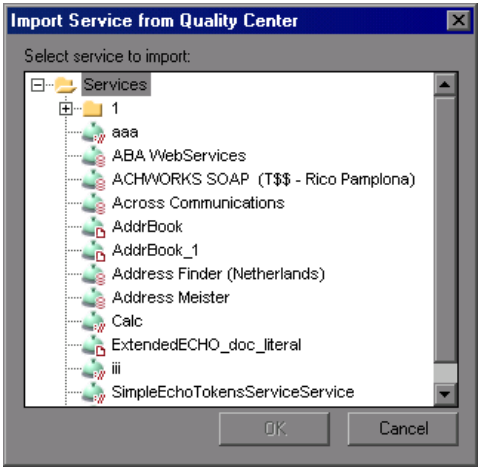
Choosing a Service from Quality Center

HP Quality Center with the Service Test Management add-on, integrates with Service Test. This integration allows you to store service entries and tests in Quality Center. You can also create and organize services according to your test requirements and test plan.

To specify a service from Quality Center:

1. In the Import Service dialog box, choose **Quality Center**.
2. If you have not yet connected to your Quality Center project, click Quality Center Connection to open the connection dialog box. For information on opening a Quality Center connection, see "Connecting to and Disconnecting from Quality Center" on page 276.

3. Click the **Browse** button to view the list of service entries saved in Quality Center.
4. Double-click on a service in the list to import it.



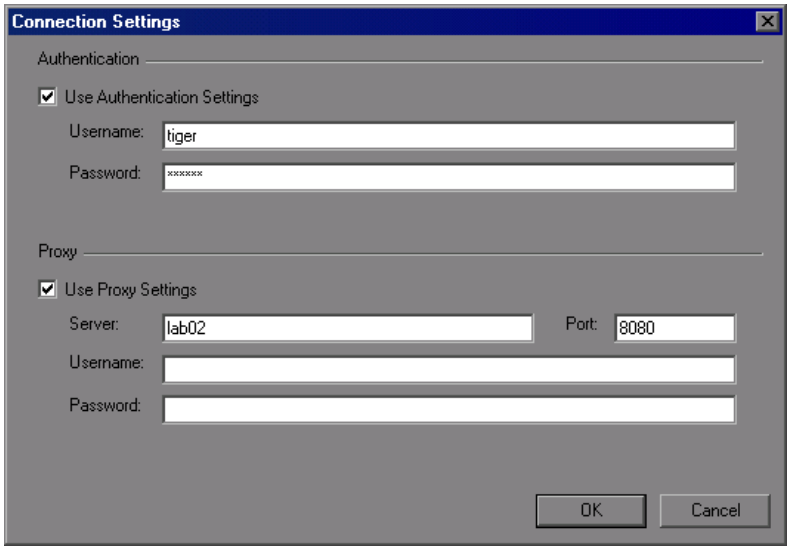
Specifying WSDL Connection Settings

Service Test supports the importing of WSDLs using authentication and WSDLs accessed through proxy servers.

Once you enter the security or proxy information, it remains with the WSDL, visible through the **Connection Settings** tab in the Service Management dialog box. If you enable the **Keep up to date** option to allow automatic synchronization, Service Test accesses the WSDL at its source using the authentication or proxy server settings.

To specify authentication or proxy information for importing:

1. Open the Import Service dialog box as you normally would, either with a new Web Service call, recording, or Traffic Analysis.
2. Select either the **URL** or **UDDI** option and specify a URL of the service to be imported.
3. In the Import Service dialog box, click the **Connection Settings** button.



4. In the Advanced Connections dialog box, select the desired option: **Use Security Settings** or **Use Proxy Server**.

5. Specify the authentication details, and, for a proxy server, the name and port of the server. If you attempt to import the secure service before specifying the necessary credentials, Service Test prompts you to enter the information.
6. To update or modify the security settings, open the Service Manager and select the appropriate service in the left pane. Click the **Connection Settings** tab. Edit the required fields and click **OK**.

Deleting Services

You can delete service entries from the Service Management dialog box, when they are no longer required. If a service was updated, you can synchronize the WSDL from the source—you don't need to delete it and reimport the service.

Before deleting a service, make sure that it is not required for your script. If you created a script based on a specific service and you then attempt to delete it, Service Test warns you that the deletion may affect your script. Deletions cannot be undone.

To delete a service, select it from the list of services and click the **Delete** button.

Comparing WSDL Files

When you import a WSDL file, Service Test makes a working copy and saves it along with the script. This saves resources and enables a more scalable and stable environment.

It is possible, however, that by the time you run the script, the original WSDL file will have changed. If you run the script, the test results may be innacurate and the script may no longer be functional. Therefore, before replaying a Web Services script that was created at an earlier date, you should run a comparison test on the WSDL file.

Service Test provides a comparison tool which compares the local working copy of the WSDL file with the original file on the file system or Web server.

If the differences are significant, you can update the WSDL from the original copy using the **Synchronize** option in the Service Management dialog box.

Service Test also has a general utility that allows you to compare any two XML files. For more information, see "Comparing XML Files" on page 332.

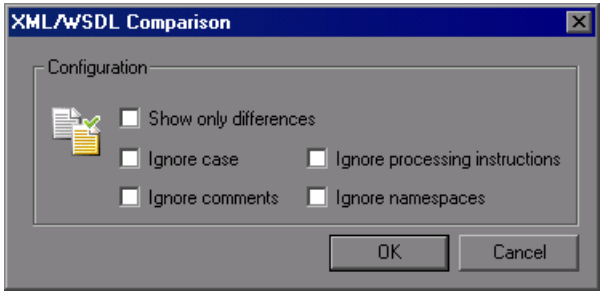
Setting WSDL/XML Comparison Options

Service Test offers the following options when comparing the local and global copies of the WSDL documents, or the revisions of an XML file:

- **Show only differences.** Show only lines with differences. Do not show the entire document.
- **Ignore case.** Ignore case differences between the texts.
- **Ignore comments.** Ignore all comments in the texts.
- **Ignore processing Instructions.** Ignore all texts with processing instructions.
- **Ignore namespaces.** Ignore all namespace differences.

To configure the comparison options:

1. Configure the comparison settings. Choose **SOA Tools** > **SOA Settings** > **XML/WSDL Compare.** The WSDL Operations Options dialog box opens. Select the desired options.



2. Click **OK**.

Note: The comparison option settings apply to both WSDL comparisons from within the Service Management window, and for XML comparisons accessed from the **Tools** menu.

Comparison Reports

Service Test lists the differences between the files in a Comparison report.

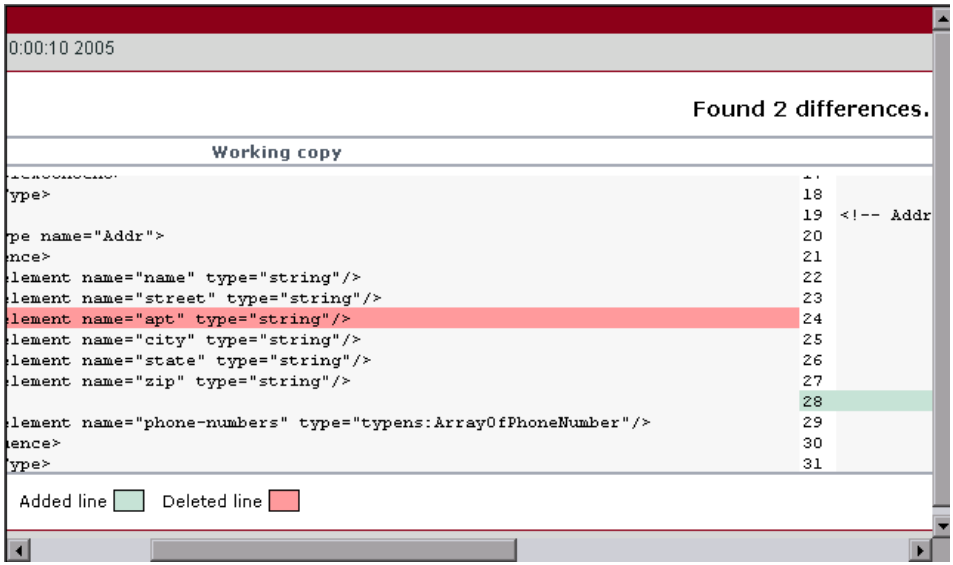
In WSDL Comparison reports, there are two columns— **Working Copy** and **Original File.** The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

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 and line 28 was added.

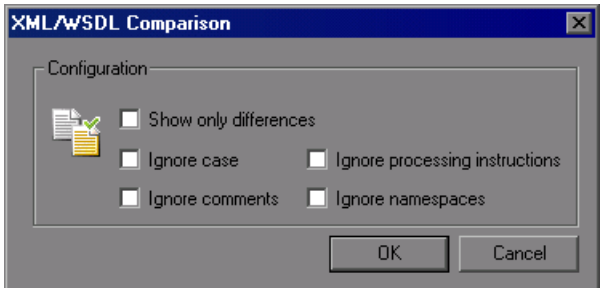


Running a WSDL Comparison

After running a file comparison, you can decide whether to ignore the changes, if they exist, or reload the WSDL file.

To compare WSDL files:

1 Configure the comparison settings. Choose **SOA Tools** > **SOA Settings** > **XML/WSDL Compare.** The WSDL Operations Options dialog box opens. Select the desired options.



2 Open the Service Management window. Choose **SOA Tools** > **Service Management** or click the **Manage Services** toolbar button

3 Select the service upon which you want to perform a comparison. You can only run the comparison on one service at a time.

4 Click **Compare**. The WSDL Comparison Report opens.

5 Scroll down through the file to locate the differences.

If you find differences between the two files and you want to update Service Test's working copy of the WSDL file, click on the WSDL file in the tree in the left pane. Select **Refresh file from global copy** from the right-click menu. This copies the current version of the WSDL into the script's WSDL directory.

6 To close the WSDL Comparison Report window, choose **File** > **Exit**.

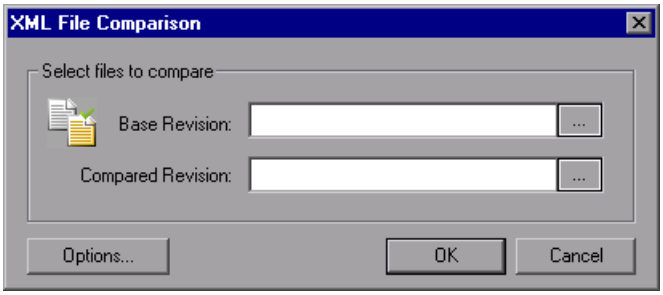
Comparing XML Files

Service Test provides a utility that lets you compare two XML files.

You can specify what differences to ignore, such as case or comments. For additional information about the comparison options, see "Setting WSDL/XML Comparison Options" on page 329.

To compare two XML files:

1 Choose **Tools** > **Compare XML Files**. The XML File Comparison dialog box opens.



2 Click the Browse button to the right of the **Base Revision** box to locate the original XML file.

3 Click the Browse button to the right of the **Compared Revision** box to locate the newer XML file.

4 Click **OK**. Service Test opens the XML Comparison Report window.

For information about the Comparison report, see "Comparison Reports" on page 330.

Performing WS-I Validation

WS-I (Web Services Interoperability) is an organization that created standards for Web Services, to promote compatibility across platforms, operating systems, and programming languages. For more information about WS-I, refer to http://ws-i.org.

If your Web Service is WS-I compliant, it means that it meets the WS-I standards and it should be suitable for multiple platforms and languages.

To claim that your Web Service is WS-I compliant, you need to validate it using a WS-I testing tool, distributed by the WS-I organization. Service Test integrates with the testing tool after you install it.

Download the tool that matches your platform: C# with .NET or Sun Java. The recommended tool for use with Service Test is Java. After you download the zip file, extract the files to a local drive, maintaining the original directory structure, **wsi-test-tools**.

You can download the tools from WS-I's Web site:

- **C#.** http://www.ws-i.org/Testing/Tools/2005/06/WSI_Test_CS_Final_1.1.zip
- **Java.** http://www.ws-i.org/Testing/Tools/2005/06/WSI_Test_Java_Final_1.1.zip

If you define an environment variable called **WSI_HOME** with a value of the path of **wsi-test-tools**, Service Test will recognize the path and automatically activate WS-I validation for WSDL. If you do not define an environment variable, you can manually enable WS-I validation and browse to the location of the WS-I testing tool.

Service Test supports WS-I Validation in several areas:

- **Service Management**. Manually validate one or more WSDLs from the Service Management window. For more information, see "Validating WSDLs" on page 335.
- **WSDL WS-I Validation Calls.** Insert validation steps into your script to automatically validate the WSDL every time your run the script. For more information, see "Adding Validation Calls" on page 336.
- **SOAP validation per step.** Validates the SOAP message for a specific step. For more information, see "Validating SOAP Messages" on page 340.
- **SOAP validation per iteration.** Validates the SOAP message for the entire iteration. For more information, see "Validating SOAP Messages" on page 340.

Note: Service Test performs WS-I validation against the original WSDL document—not against the local copy.

Configuring WS-I Validation

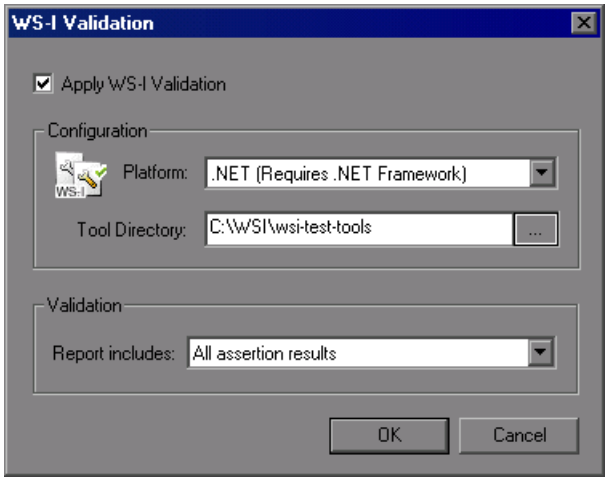
Service Test lets you validate a WSDL or the SOAP message sent by the service during record or replay.

Before performing validation, make sure that WS-I validation is enabled and configured according to your requirements:

To enable WS-I validation:

1. Choose **SOA Tools** > **SOA Settings** > **WS-I validation**. Alternatively, click the **WS-I Validation** button in the Service Management window. If this is the first time you are doing a validation, the WSI-validation box opens.

2 Select **Apply WS-I Validation**.



3 Choose the **platform** of your Web Service: Microsoft .NET, which requires the .NET Framework, or Java (Sun Java).

4 Specify the location of the WS-I validation tool directory, **wsi-testing-tool**.

5 Indicate which messages to include in the report:

- **All assertion results**. Show all results.
- **Assertions with the result "Failed"**. Show only the assertions with a "Failed" result status.
- **Assertions with a Result different than "Passed"**. Show the assertions that did not have a "Passed" result status.

6 Click **OK**.

Validating WSDLs

You can validate your WSDL file before or after creating a script.

To validate a WSDL file:

1 Open the Service Management window. Choose **SOA Tools** > **Service Management** or click the Manage Services toolbar button

2 Select the services you want to validate. Use the **Ctrl** button to select multiple services.

3 Click **WS-I Validation**. If this is the first time you are doing a validation, you need to configure the WS-I settings as described in "Configuring WS-I Validation" on page 334.

4 Click **OK**. The WS-I Validation Report opens.

Adding Validation Calls

You can manually add validation call to check a specific service every time you replay the script. It is recommended that you make a separate test script to perform the validations since if the validation fails, the script execution stops.

When you run your script, Service Test runs the validation and submits a message to the Output log indicating whether the WSDL is WS-I compliant. It also provides the location of the validation report.

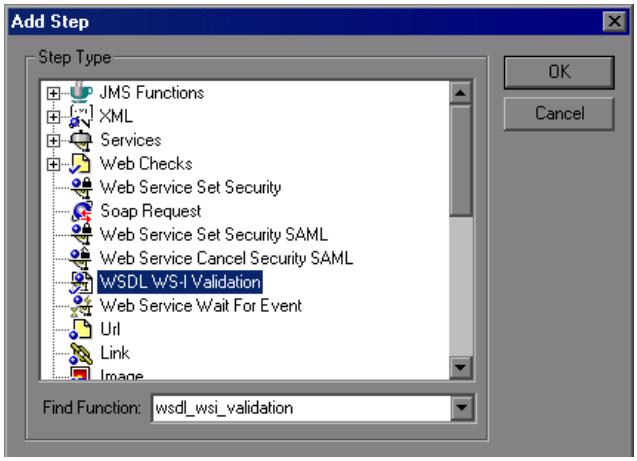
```
Action.c(33): WSDL WS-I validation for service "Calc" started
Action.c(33): Error: The service is not WS-I compliant
Action.c(33): For more information, see report in "C:\Program
    Files\Mercury\Mercury SOA Tester\scripts\My Test
   3\WSDL\WSIValidationReport_Calc_CalcSoapBinding.xml"
```

If you are running several iterations of the script, it is recommended that you place the validation step in the **init** section. This will allow Service Test to validate the service only once, instead of upon every iteration.

To manually add a validation step to your script:

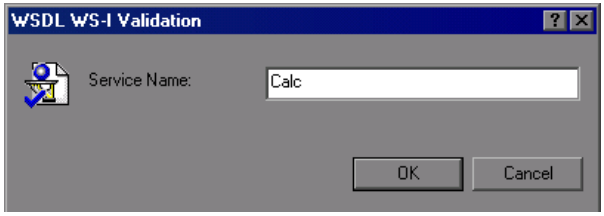
1 In Tree view, click at the appropriate location in your script.

2 Choose **Insert** > **New Step** to open the Add Step dialog box.



3 Select **WSDL WSI Validation** and click **OK**.

4 Provide the name of the service you want to validate.



In Script view, you can add **wsdl_wsi_validation** function. For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

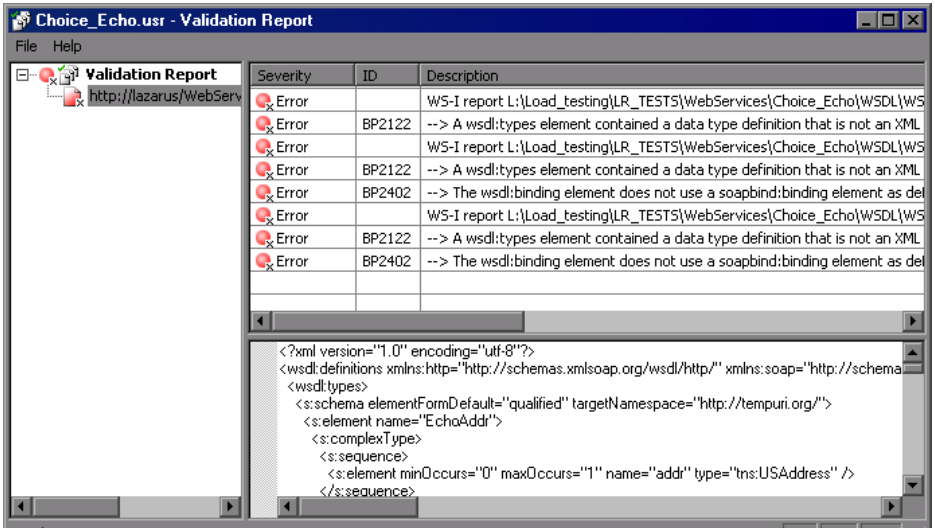
Validation Reports

WSDL Validation Reports provide information about your WSDL file. A green icon in the **Severity** column implies that there are no problems, yellow indicates warnings, and red marks an error.

You can view several types of reports to help you understand compliancy issues: WSDL Validation Report and the native WS-I report.

WSDL Validation Report

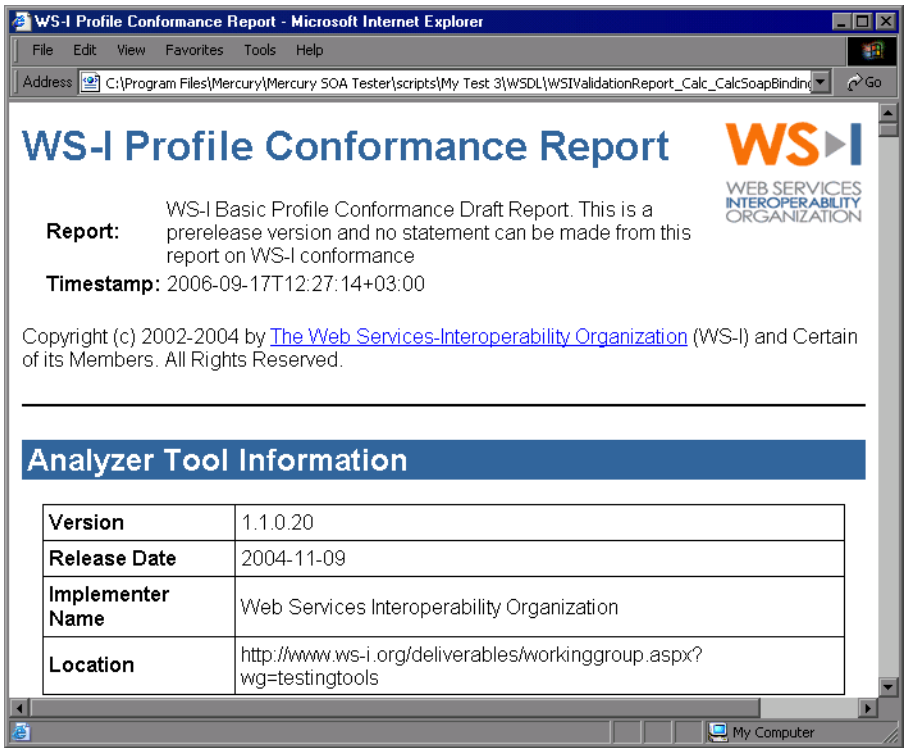
If there is an error in an WSDL file, Service Test displays its details in the right pane.



To save the report to an HTML file, choose **File** > **Export as HTML**.

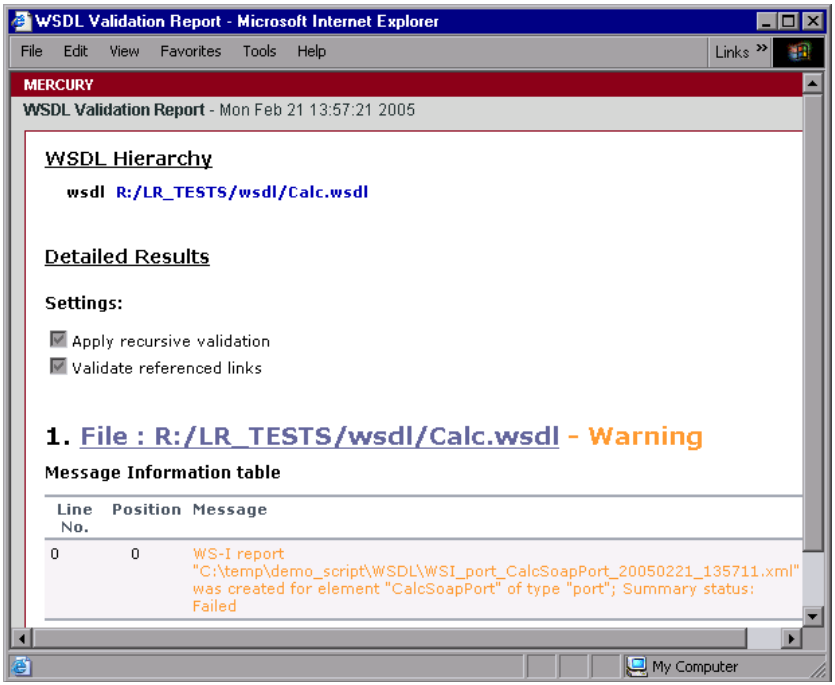
Native WS-I Reports

Native WS-I reports are WS-I basic profile conformance reports that provide details as to why the WSDL complies or does not comply with the WS-I standards.



To view the native WS-I report, select the WS-I error in the Validation report and choose **Open WS-I Report** from the right-click menu. A browser opens with the native WS-I report describing the error in detail.

You can also save the report to an HTML file for later viewing.



To create and save an HTML summary report of the validation:

1 Choose **File** > **Export to HTML**. Service Test opens a browser with an HTML report of the Validation results.

2 Choose **File** > **Save As** from the browser window to save the HTML file.

Validating SOAP Messages

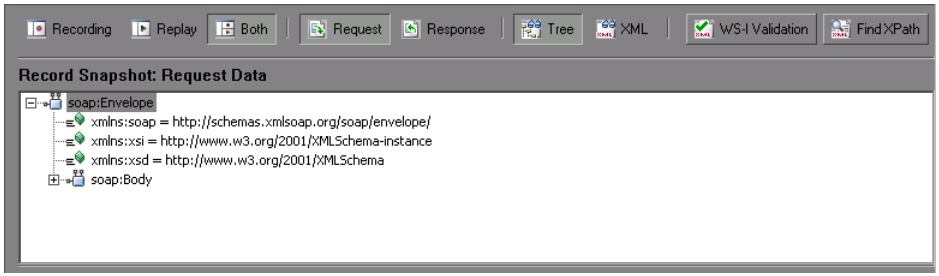
Service Test lets you validate the SOAP messages of the various responses.

To validate the SOAP message for a specific step:

1 In Tree view, select the s

2 tep you want to validate.

3 Select the snapshot tab.



4 Click the **WS-I Validation** button to begin the validation.

You can also validate the SOAP messages of an entire iteration—either Record or Replay.

- ➤ To validate the recorded snapshot, choose **SOA Tools** > **SOAP Validation** > **Record Validation**.
- ➤ To validate the current replay snapshot, choose **SOA Tools** > **SOAP Validation** > **Replay Validation**.

Service Test opens the Validation report with the validation data. For more information, see “Validating WSDLs” on page 335.

Viewing WSDL Files

The Service Management window lets you view the WSDL in your default browser.

To view a WSDL:

1 Select it in the left pane.

2 Click the View WSDL button in the Service Management window.

22

Adding Content to Web Services Scripts

You use Service Test to create a script to test your Web Services through recording, manually adding calls, or analyzing server traffic.

The following information only applies to Web Services and SOA Vuser scripts.

About Adding Content to Web Services Scripts

Web Services scripts let you test your environment by emulating Web Service clients.

After creating an empty Web Services script, as described in "Understanding the SOA Test Types" on page 307, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

To create scripts automatically, run the SOA Test Generator using the wizard to add content. For more information, see Chapter 23, "Using the SOA Test Generator."

For information on creating scripts by	See
recording	"Recording a Web Services Script" below
manually inserting Web Service calls	"Adding New Web Service Calls" on page 350
importing SOAP requests	"Importing SOAP Requests" on page 352
analyzing server traffic	Chapter 24, "Creating Server Traffic Scripts"
running the SOA Test Generator	Chapter 23, "Using the SOA Test Generator."

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.

Specify Services for Recording

When you record an application, you can record it with or without a Web Service WSDL file. It is recommended to record with a WSDL when possible.

If you include a WSDL file, Service Test allows you to create a script by selecting the desired methods and entering values for their arguments. Service Test creates a descriptive script that can easily be updated when there are changes in the WSDL.

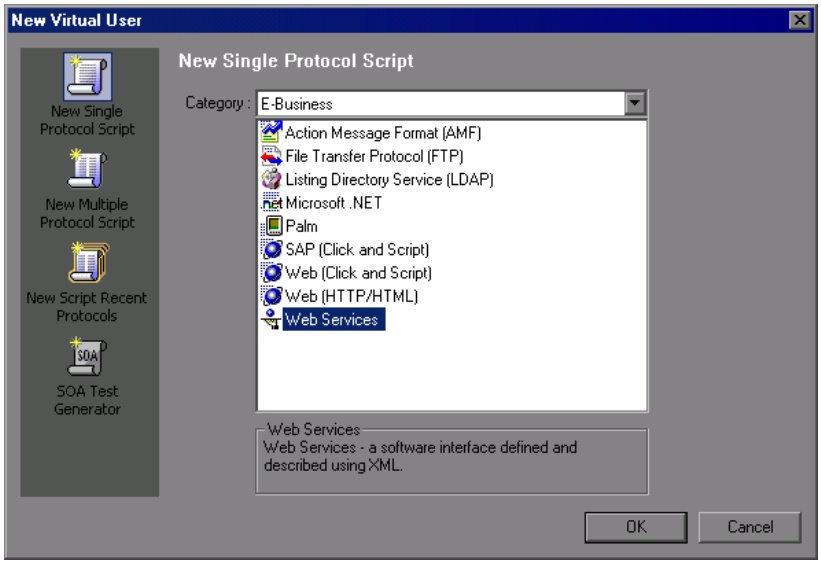
If you do not specify a WSDL file (not recommended), Service Test creates a test with SOAP requests instead of Web Service call steps. If you create a script without importing a service, Service Test creates a **soap_request** step whose arguments are difficult to maintain.

To create Web Services script through recording:

1 **Create an empty script.**

Choose **File** > **New** to open the New Virtual User dialog box.

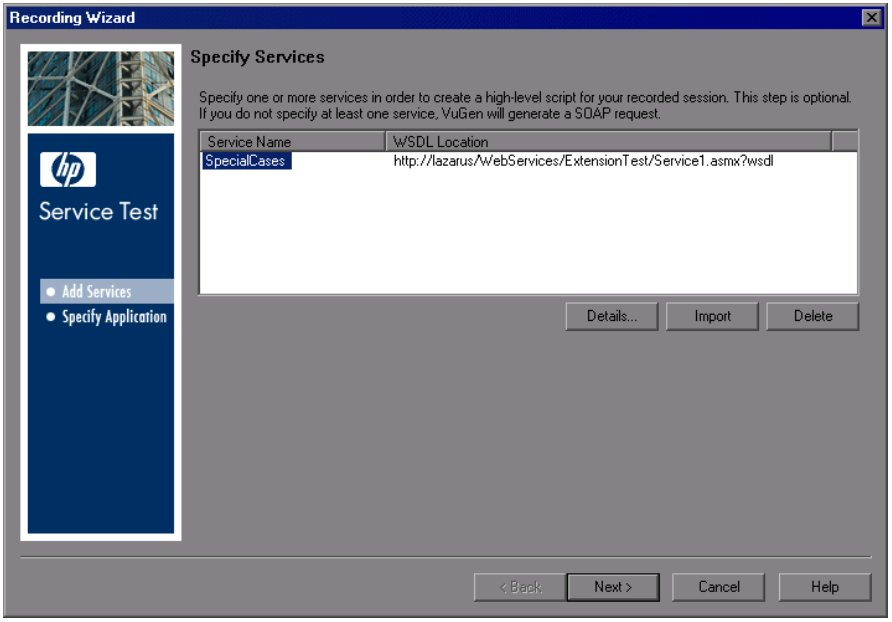
For a single protocol script, click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the E-Business category. Click **OK**.



If you need to record several different protocols, such as Web Services and Web, click **New Multiple Protocol Script** in the left pane and specify the desired protocols. Click **OK**.

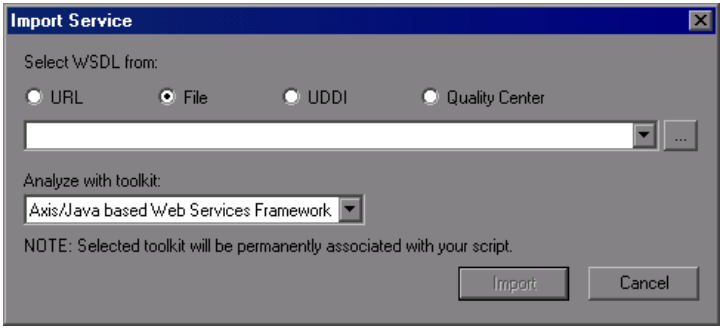
2 Begin the recording process.

Click the **Start Record** button or Ctrl+R to open the Specify Services screen.



3 Add a service to the list.

To produce a high-level Web Service script, add one or more services using the **Import** button. If the WSDL of the recorded Web Service is available, it is recommended to import it here. If you create a script without importing a service, Service Test creates a **soap_request** step. The Import Service dialog box opens.



a Choose a source and location for the WSDL.

b Select a toolkit. The toolkit you choose is permanently associated with the script. For more information, see "Importing Services" on page 322.

c Click **Import**.

Repeat the above step for each WSDL you want to import.

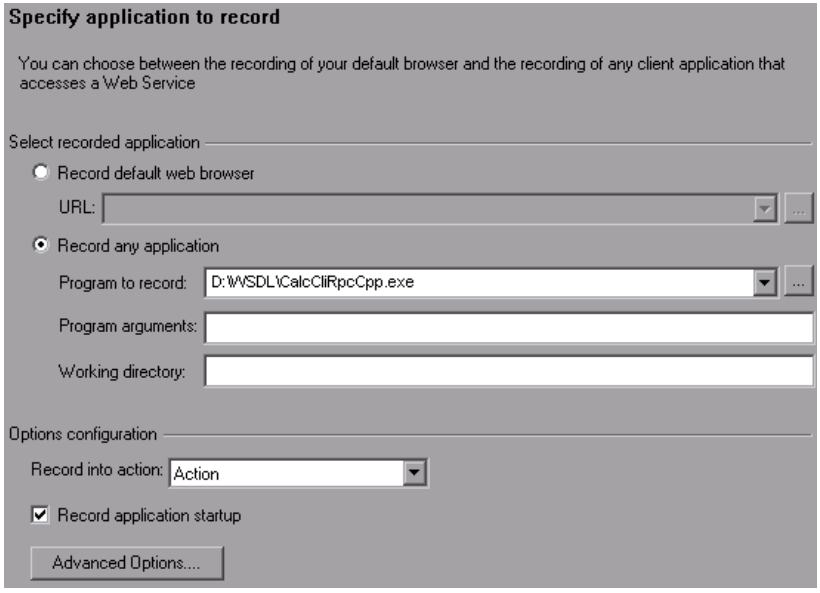
4 Enter details for the Web Service.

Click **Details** to open the Service Manager window and view the details of the Web Services that you added. For more information, see Chapter 25, "Working in the Web Service Call View."

In the next step, you choose an application to record.

Selecting an Application to Record

In this screen you specify the application to record. You can record a browser session or client application.



1 Specify the recording details.

- **Record default Web Browser.** Records the actions of the default Web browser, beginning with the specified URL. Select this option where you access the Web Service through a Web-based UI.
- **Record any application.** Records the actions of a your client application. Specify or browse for the path in the **Program to record** box. Specify any relevant arguments and working directories.

2 Configure options.

- **Record into action.** The action in which to generate the code. If there are startup procedures that you do not need to repeat, place them in the **vuser_init** section. During recording you can switch to another section, such as **Action**.
- **Record application startup.** Records the application startup as part of the script. If you want to begin recording at a specific point, not including the startup, clear this check box.
- **Advanced Options.** Opens the Recording Options dialog box, allowing you to customize the way in which the script is generated. For more information, see Chapter 6, "Setting Script Generation Preferences."

3 Click Finish.

Service Test opens the application and begins recording. Perform the desired actions within your application and then press the **Stop** button on the floating toolbar. Service Test generates **web_service_call** functions, or **soap_request** functions if you did not import a WSDL.

After recording, you can enhance your script with additional service calls and parameterization.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 308.

Viewing the Workflow

When adding script content through recording or Analyzing traffic, Service Test's workflow guides you through the stages of preparing a script. By clicking in the **Tasks** pane, you can read about the steps for creating a script, view information about your recording, and verify the replay. Use the **Back** and **Next** buttons to navigate between screens.

If you do not see the Workflow screen, click the **Tasks** button on the toolbar to open the Tasks pane, and select a task.

Create Script

The Create Script screen provides several guidelines for creating a Web Services script. It also provides a link for opening the Web Services wizard.

- **Before You Start.** Describes what you should know before you begin.
- **About Script Creation.** Describes the stages of script creation.
- **Actions.** Describes script sections and why they are important.

There are two action-related links:

- **Start Recording.** Opens the Start Recording dialog box where you provide information about the application to record.
- **Analyze Traffic.** Lets you analyze traffic captured over the network to create a script that emulates a server.

Creation Summary

After you create a script, the Creation Summary screen provides information about the recording or script generation.

- **Protocols.** Lists which protocols were used during the script creation.
- **Actions.** Describes into which sections actions were recorded or imported.

It also provides links that allow you to modify the script:

- **Add a web_service_call statement.** Lets you manually add a **web_service_call** function to your script by specifying a service, operation, and argument values.
- **Manage the services.** Opens the Service Repository window with a list of all of the services that are available to the script and their properties.
- **Compare XML files.** Allows you to compare two versions of an XML file. This is useful for comparing WSDL files and determining if there was a change since your originally imported it into the script.

For information about the remainder of the workflow, see Chapter 3, "Viewing the VuGen Workflow."

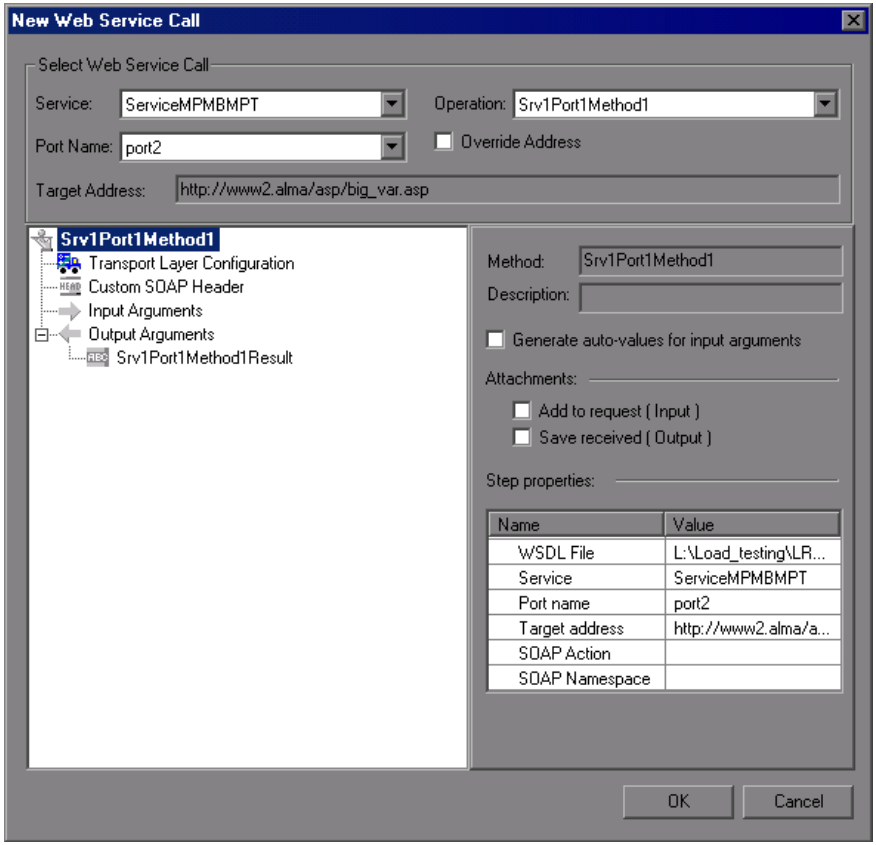
Adding New Web Service Calls

You can add new Web Service call functions in both Tree and Script views.

To add a Web Service call:

1. Click the cursor at the desired location in your script.
2. Click the **Add Service Call** button. The New Web Service Call dialog box opens. Select the desired **Service**. If this is a new script and you did not yet import a WSDL, do so at this point. For more information, see "Importing Services" on page 322.

3 Select an **Operation**. For services using multiple ports, select a **Port Name** for the operation. This lets you differentiate between identical operations performed on separate ports.



4 To specify a target address other than the default for the active port, select **Override address** and enter the address.

5 To provide sample input values for the service, click on the highest level node (the service name) and select **Generate auto-values for input arguments**. Service Test adds sample values, and places an arrow before each of the arguments, to indicate that it is using auto-values.

To parameterize the input arguments of the operation, see “Setting Properties for XML Parameters” on page 180.

6. Select the Transport Layer Configuration tab to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see "Configuring the Transport Layer" on page 418.
7. Click on each of the nodes and specify your preferences for argument values. For more information, see Chapter 25, "Working in the Web Service Call View."
8. To add an attachment to an input argument, or to specify a parameter to store output arguments, select the operation's main node and make the appropriate selection. For more information, see "Attachments" on page 394.

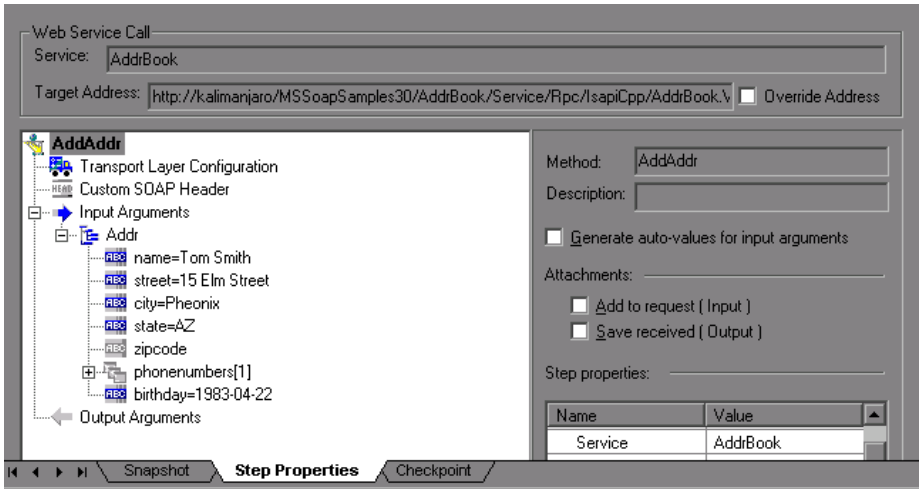
Importing SOAP Requests

Service Test lets you create Web service calls from SOAP files. If you have a SOAP request file, you can load it directly into your script. Service Test 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.

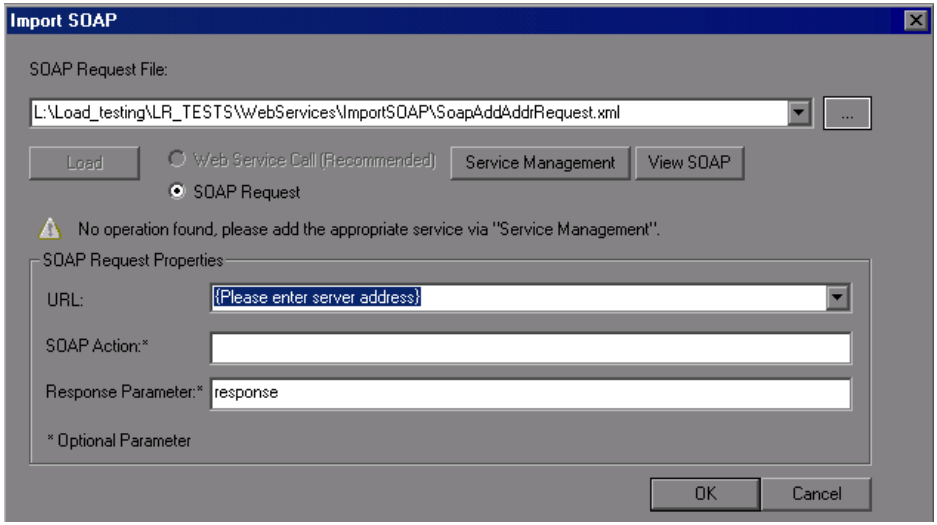
For example, suppose you have a SOAP request with the following elements:

```
- <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, Service Test imports all of the values to the Web Service call:



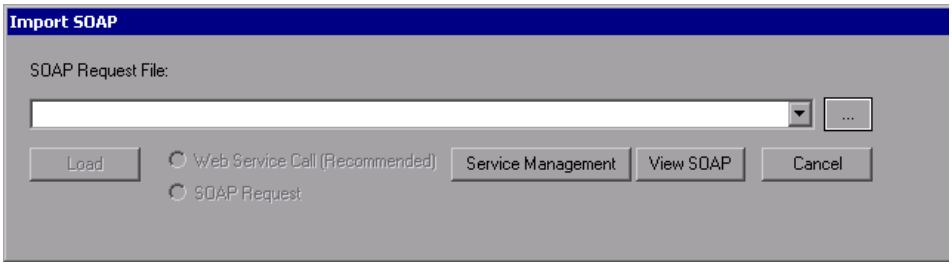
To create a new Web Service call based on a SOAP request, you must first import a WSDL file. If a WSDL is not available, or if you want to send the SOAP traffic directly, you can create a SOAP Request step. You specify the URL of the server, the SOAP action, and the response parameter.



In Script view, the SOAP Request step appears as a **soap_request** function, described in the *Online Function Reference* (**Help** > **Function Reference**).

To import a SOAP request:

1 Click the **Import SOAP** button or choose **SOA Tools** > **SOAP Import**.



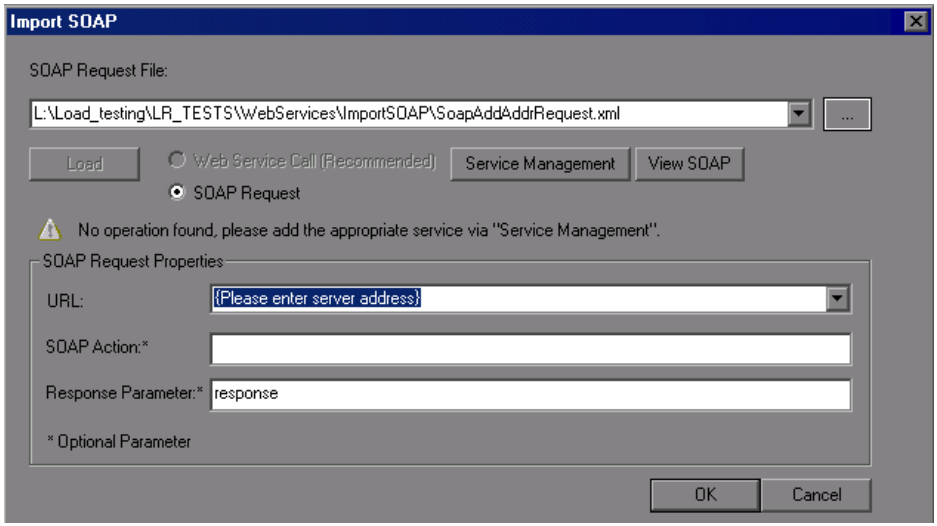
2 Browse for the XML file that represents your SOAP request.

3 Choose 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 import a WSDL, click **Service Management** and then click the **Import** button. To view the SOAP before loading it, click **View SOAP**.

4 Click **Load**. Service Test loads the XML element values.

For a Web Service Call, set the properties for the Service call as described in "Understanding Web Service Call Properties" on page 385.

For a SOAP request, provide the URL and the other relevant parameters.



5 For a Web Service Call, if there are multiple services with same operation (method) names, you need to choose the service whose SOAP traffic you want to import. For information about additional properties, see "Understanding Web Service Call Properties" on page 385.

6 Click **OK** to generate the new step within your script.

7 Set checkpoints and replay the step. For more information, see "Setting Checkpoints" on page 409.

Using Your Script

After you create scripts, you can manage them in one of the following ways:

➤ **Service Test Management.** An add-on for HP Quality Center that lets you manage SOA testing by allowing you to import, store and define services in Quality Center. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management. For more information, see "Working with Service Test Management" on page 361.

➤ **BPT.** Business Process Testing lets you combine several scripts to form a complete business process. For more information, see "BPT (Business Process Testing)".

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 Chapter 27, "Running SOA/Web Services Scripts."
- **Load Testing.** Integrate the script into a LoadRunner Controller scenario to test its performance under load. For more information, see the *HP LoadRunner Controller* or *Performance Center* documentation.
- **Production Testing.** Check your Web service's performance over time through a Business Process Monitor profile. For more information, refer to the *HP Business Availability Center* documentation.

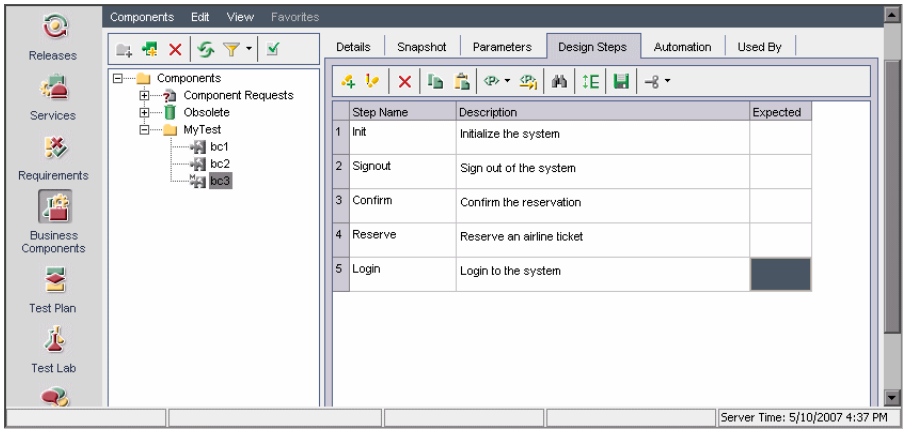
BPT (Business Process Testing)

BPT (Business Process Testing) is a methodology in which several tests are combined to create a complete business process. The BPT user composes a complete test by combining a series of test components with data flow between them.

Components are comprised of steps. For example, a login component's first step may be to open the application. Its second step could be entering a user name. Its third step could be entering a password, and its fourth step could be clicking the **Enter** button.

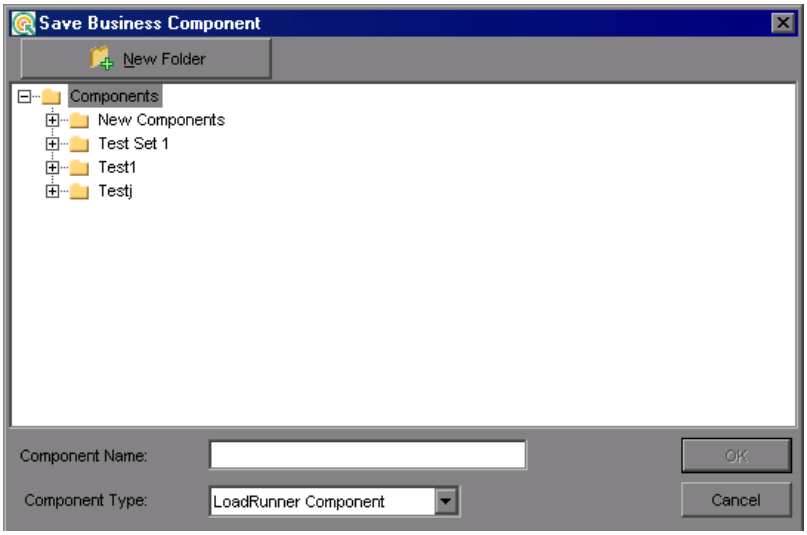
You can create business process test components from within Quality Center or through HP Service Test.

In Quality Center, non-technical SMEs (Subject Matter Experts) define design steps that are required for the test.

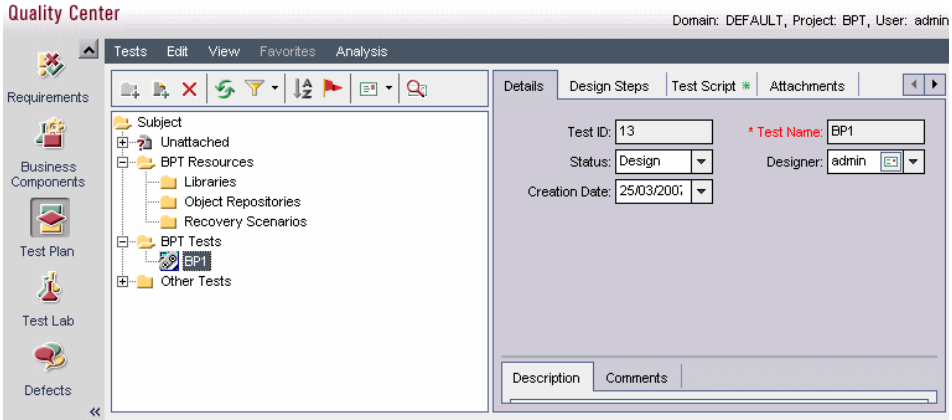


After these are defined, the technical engineer creates a script through Service Test that performs the desired actions. For more information about creating Business Components from within Quality Center, see the *Business Process Testing user's guide*.

You create components in Service Test by recording a session with your application or manually editing a script. You can add checkpoints, parameterize selected items, and enhance the component with flow statements and other testing functions. You then save the component to a project in Quality Center.



A Subject Matter Expert using Business Process Testing in Quality Center combines your saved components into one or more business process tests, which are used to check that the application behaves as expected.



When you create parameters for your business component in Quality Center, they will be available in Service Test's parameter list.

Some of the advantages of working with a BPT module over individual scripts are:

- Enables less-technical subject matter experts to create tests
- Enables structured automated testing
- Reduces the duplication of effort when combining manual tests with automatic scripts
- Allows component reusability to speed-up the automation process
- Provides the ability to pass parameters from one step to another within your business process. You can save the output of a step to a parameter and use it as an input value for subsequent steps.
- Simplifies on-going test maintenance
- Minimizes time-to-test

For more information about creating a BPT components in Quality Center, see the *Business Process Testing User's Guide*.

Creating Business Components in Service Test

You can create a new business component in Service Test and add content through recording or any other means as described in this section. You can also save existing scripts as business components.

To create a Business Component in Service Test:

1. Open Service Test and connect to the Quality Center server. Select **Tools** > **Quality Center Connection** and specify the connection details.
2. Select **File** > **Business Component** > **New**. The New Business Component dialog box opens.
3. Create a new Web Services script and add script content by recording, importing SOAP, or by manually adding Web Service calls. For more information, see Chapter 19, "Understanding the SOA Test Types."

4. Parameterize the desired arguments. If you want to share the parameter between business components in Quality Center, make sure to create a **BPT** type parameter. For more information, see "BPT Type Parameters" on page 152.
5. Save the script to the desired location in Quality Center. When you create a business component in Service Test, it is regarded as an automated test in Quality Center.
6. To convert an existing Vuser script to a Business Component, select **File** > **Business Component** > **Save as Business Component** and save the script to the desired location in Quality Center.

Creating Components in Quality Center

The following section describes briefly how to create a business component in Quality Center and automate it to be compatible with Service Test. For complete information about working with Business Components in Quality Center, see the *Business Process Testing User's Guide*.

To create a Service-Test type automated business component in Quality Center:

1. Select the **Business Components** module in the left pane and choose **Component** > **New Component** to create a new business component.
2. Select the **Design Steps** tab, create new manual steps, and define parameters (optional).
3. Automate the business component. In the **Design Steps** tab, choose **Automate Component** (rightmost button) > **Service Test.**

Editing Automated Components in Service Test

You can also use Service Test to edit a SERVICE-TEST Automated component, that was created and automated in Quality Center. You can open the business component in Service Test or directly from Quality Center.

To open and edit an automated business component in Service Test:

1. Open Service Test and connect to the Quality Center server. Select **Tools** > **Quality Center Connection** and specify the connection details.

2 Select **File** > **Business Component** > **Open**. The Open Business Component dialog box opens.

3 Specify a script to open.

4 Edit the script, set parameters and add content as you would with any other script. For more information, see "About Adding Content to Web Services Scripts" on page 344.

To open and edit an automated business component from Quality Center:

1 Select the **Business Components** module in Quality Center's left pane and select a business component in the tree.

2 Select the **Automation** tab.

3 Click **Launch**. Service Test opens with the script.

4 Edit the script, set parameters and add content as you would with any other script. For more information, see "About Adding Content to Web Services Scripts" on page 344.

For more information about working with Business Components in Quality Center, see the *Business Process Testing User's Guide*.

Working with Service Test Management

HP Quality Center is a Web-based application for test management. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management.

The **Service Test Management** add-on for Quality Center, lets you manage SOA testing by allowing you to import, store and define services in Quality Center.

The Service Test Management integration lets you:

- **Store Web Services.** You can store and organize Web Services in Quality Center for use within Service Test.
- **Write Service Test scripts.** You can create scripts based on the services stored in Quality Center, while maintaining up-to-date WSDLs throughout the life-cycle of the service and the script.

- **Compose a Business Process Test.** You can create a BPT (Business Process Test) in Quality Center based on services defined through Service Test Management.

Service Test Management also integrates with HP's Systinet Registry, to create test requirements and plans. Once the services are imported, Service Test Management identifies any changes to the services and automatically generates the necessary test cases that need to be run.

Using the **Service Test Management** add-on for Quality Center, groups throughout your organization can contribute to the quality process in the following ways:

- Business analysts define application requirements and testing objectives.
- Test managers and project leads design test plans and develop test cases.
- Test managers automatically create QA testing requirements and test assets for SOA services and environments.
- Test automation engineers create automated scripts and store them in the repository.
- QA testers run manual and automated tests, report execution results, and enter defects.
- Developers review and fix defects logged into the database.
- Project managers can export test and resource data in various reports, or in native Microsoft Excel for analysis.
- Product managers decide whether an application is ready to be released.
- QA analysts can auto-generate test asset documentation in Microsoft Word format.

For more information about the integration, refer to the *HP Service Test Management User's Guide*.

23

Using the SOA Test Generator

The SOA Test Generator automatically generates test scripts based on your requirements.

The following information only applies to Web Services Vuser scripts.

About Using the SOA Test Generator

To test your SOA environment, you can create tests manually, or use the SOA Test Generator to automatically generate scripts.

This section describes how to use the SOA Test Generator. For information on creating tests manually, see Chapter 22, "Adding Content to Web Services Scripts."

The SOA Test Generator guides you through the process of creating scripts to test your services. Through the wizard, you indicate which aspects of the service you want to test. These aspects include interoperability with different toolkits, boundary testing, and standard compliance.

After you select the testing aspects, Service Test automatically generates one or more scripts for each of the aspects.

Selecting Test Aspects

The SOA Test generation wizard helps you create scripts that test different aspects of your service. It creates a separate script for each aspect. If an aspect has sub-aspects, Service Test creates a separate script for each one of the sub-aspects.

Testing Aspects

The SOA Test Generator supports the following testing aspects:

- **Positive Testing.** Generates a full positive test for the selected services. It tests the service operations both separately and with one another.
- **Standard Compliance.** Checks the service's compliancy with industry standards such as WS-I and SOAP.
- **Service Interoperability.** Checks that the service's operations are interoperable with all supported Web Services toolkits. Note that several tests are created for this aspect.
- **Security testing.**
  - **SQL Injection.** Attempts to hack the service by injecting SQL statements that will result in an unauthorized extraction of data.
  - **Cross-site Scripting.** Attempts to hack the service by injecting code into a Web site that will disrupt its functionality.
- **Boundary Testing.** Tests the services using the negative testing technique. Based on the service's description, Service Test creates tests to manipulate data, types, parameters, and the actual SOAP message in order to test the service to its limits.
  - **Extreme Values.** Tests for extreme values of simple data types.
  - **Null values.** Provides NULL parameters to the services to verify that they are not accepted.

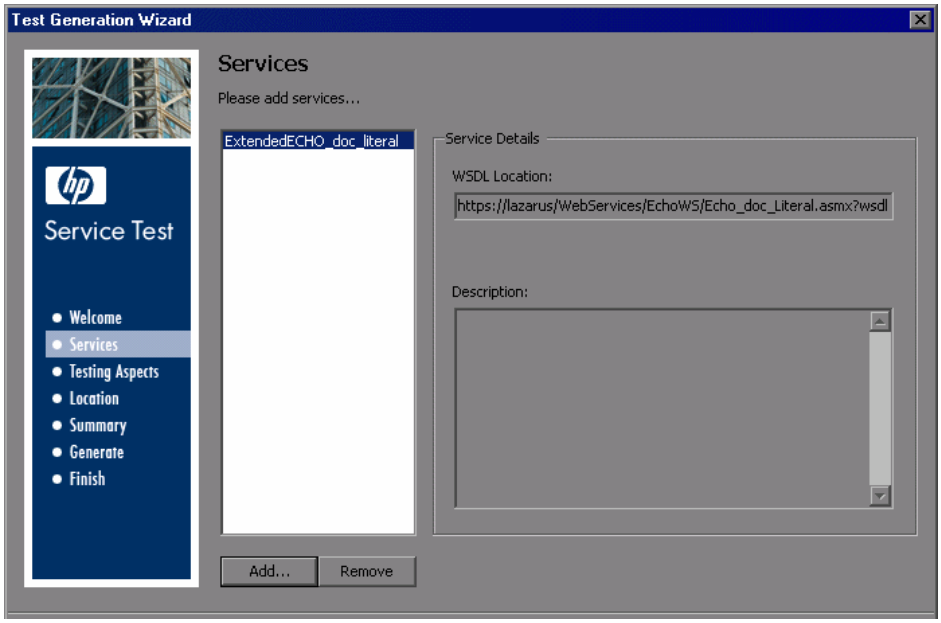
Tip: When working with Quality Center, it assigns a Pass or Fail status to each of the scripts. For more information, refer to the documentation for the Service Test Management module.

Generating Tests Automatically

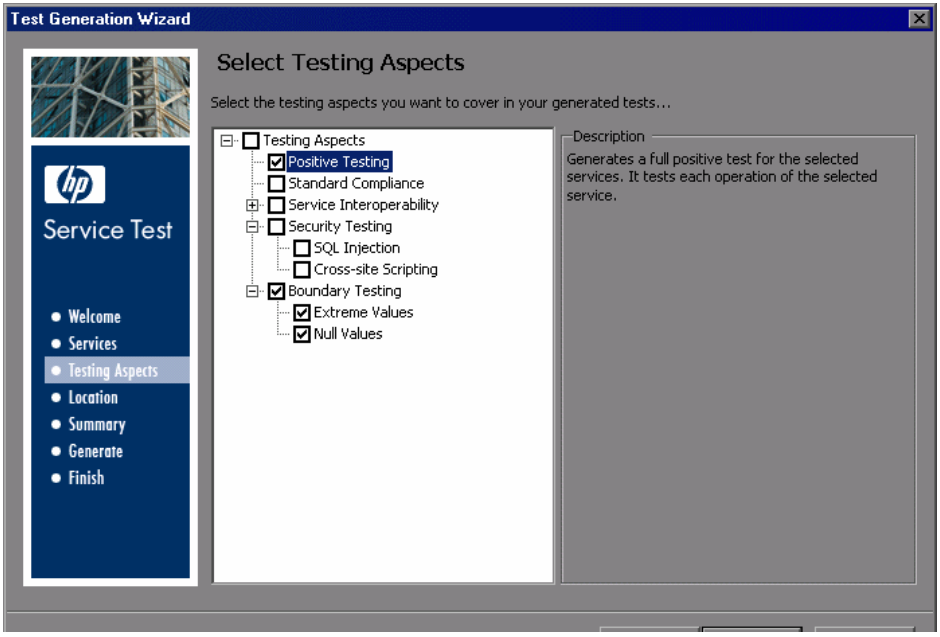
The Test Generator's wizard guides you through the steps of creating automatic tests:

To create automatic tests:

1. Choose **File** > **New** to open the New Virtual User dialog box.
2. Select **SOA Test Generator** in the left pane.
3. Make sure **Web Services** is listed in the Selected Protocols box. Click **OK**.
4. In Test Generation Wizard welcome screen, click **Next**.

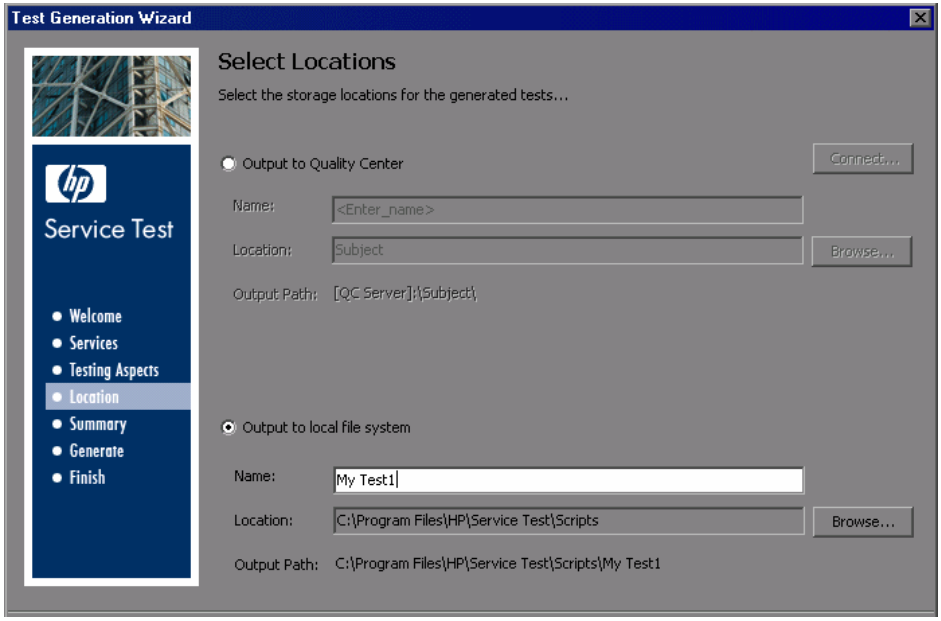


5. Add at least one service. Click **Add** to open the Import Service dialog box.
6. Specify the WSDL information. For more information, see "Importing Services" on page 322.
7. Repeat this step for all of the services you want to import. Click **Next** to select testing aspects.
8. Select the desired testing aspects and sub-aspects. For more information, see "Selecting Test Aspects" on page 364.



Click **Next** to select the tests' output location.

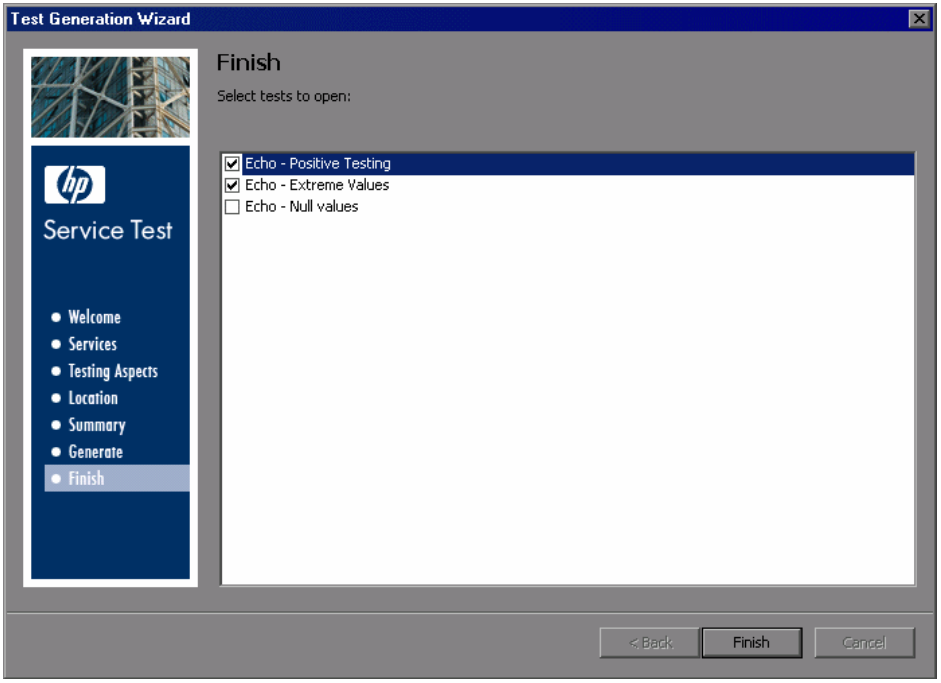
9 Specify a test name and a location for the test scripts: **Quality Center** or a **local file system**.



If you specified Quality Center, click **Connect** to log on to the server and **Browse** to locate the test node.

10 Click **Next** to display a summary of your selections: selected services, testing aspects, and output location.

11 Click **Generate** to begin creating the scripts. After Service Test completes generating the tests, it lists them in the **Finish** screen.



12 Select the scripts that you want Service Test to open and click **Finish**. The Script Generator opens the selected scripts.

After you create scripts, you can enhance them with additional service calls and parameterization. You then run the completed script in Service Test to check its functionality.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 308.

24

Creating Server Traffic Scripts

Using Service Test, you can create scripts to test your Web Service by analyzing server traffic capture files.

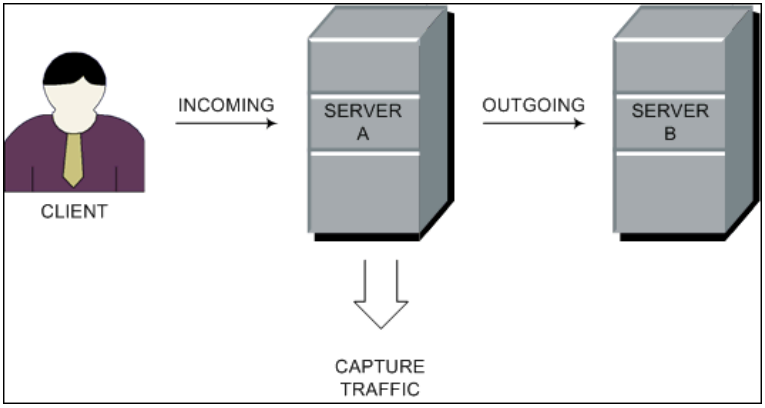
The following information only applies to Web Services/SOA Vuser scripts.

About Creating Server Traffic Scripts

The main focus when testing enterprises and complex systems, is to measure the performance from the client end. Ordinarily, Service Test 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 Service Test'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 below in Getting Started with Server Traffic Scripts.



There are two types of emulations: **Incoming traffic** 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. Service Test 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 Service Test 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**.

Getting Started with Server Traffic Scripts

The following section outlines the process of creating a script that analyzes server traffic.

1 **Create a capture file.**

Service Test uses the capture file to analyze the server traffic and emulate it. For more information, see "Generating a Capture File" on page 372.

2 **Create a new Web Services script.**

Using Service Test's main interface, you create a new Web Services script. For more information see Chapter 19, "Understanding the SOA Test Types."

3 **Specify the Services (optional, but recommended).**

To create a high-level script, import a WSDL which describes the Web Service you want to test.

4 Specify the traffic information.

Specify the location of the traffic file and whether your script will be for Incoming or Outgoing traffic. For more information, see "Specifying Traffic Information" on page 376.

5 Specify the traffic filter Recording options.

Filter options let you determine which hosts to include or exclude in your script. For more information, see "Choosing an Incoming or Outgoing Filter" on page 377.

6 Specify the SSL certificate information.

The SSL configuration lets you analyze secure traffic over HTTPS in order to generate the script. For more information, see "Providing an SSL Certificate" on page 378.

Generating a Capture File

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.

TheService Test Command Line Utility

The Service Test command line utility, **lrtcpdump**, is located in the product's **bin** directory. There is a separate utility for each of the platforms: **lrtcpdump.exe** (Windows), **lrtcpdump.hp9**, **lrtcpdump.ibm**, **lrtcpdump.linux**, and **lrtcpdump.solv4**.

To invoke the capture tool, type:

```
lrtcpdump -i<interface> -f<file>
```

where **interface** is the name of the network card whose traffic you want to capture, and **file** is the name of the capture file in which to store the information. Do not leave a space between the command line option (**i** or **f**) and the value.

To create a capture file on a Windows platform:

1. Choose **Start** > **Run**, type **cmd** and click **OK** to open a command window.
2. Drag in or enter the full path of the **lrtcpdump.exe** program located in the product's **bin** directory.
3. Provide a file name for the capture file using the following syntax:
   ```
   lrtcpdump -f <file>
   ```
   for example **lrtcpdump -fmydump.cap**.
4. **lrtcpdump** prompts you to select a network card. If there are multiple interface cards, it lists all of them. Type in the number of the interface card (1, 2, 3 etc.) and click **Enter**.
5. Perform typical actions within your application.
6. Return to the command window and click **Enter** to end the capture session.

To create a capture file on a UNIX platform:

1. Locate the appropriate **lrtcpdump** utility for your platform in the product's **bin** directory. 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.
2. Switch to the root user to run the utility.
3. Provide execution permissions. **chmod 755 lrtcpdump.<platform>**
4. On UNIX platforms, 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.
5. Run the utility with its complete syntax, specifying the interface and file name. For example, **lrtcpdump.hp9 -ietho -fmydump.cap**. The capturing of the network traffic begins.

6. Perform typical actions within your application.
7. Return to the window running **lrtcpdump** and follow the instructions on the screen to end the capture session.
8. Place the capture file on the network in a location accessible to the machine running Service Test.

An Existing Capture Tool

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.

Note: Certain utilities require additional arguments. For example, **tcpdump** requires the **-s 0** argument in order to capture the packets without truncating their data.

Creating a Basic Script from Server Traffic

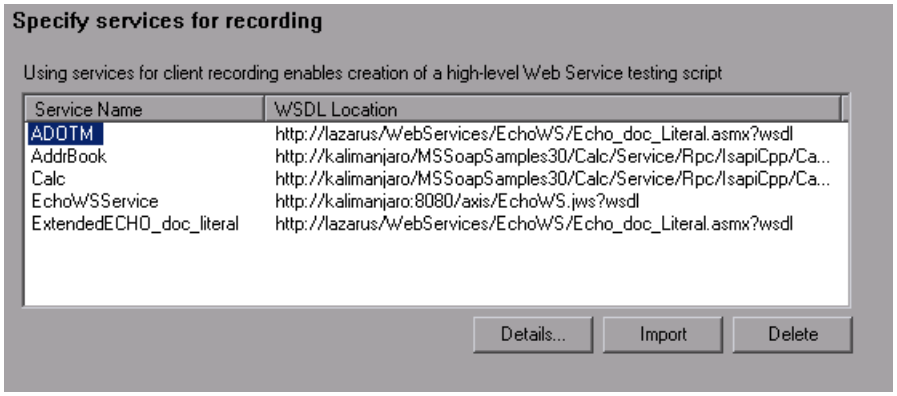
You create a script from server traffic just as you would create a recorded script.

You can optionally specify a Web Service for your script. If you specify a service, Service Test will create a script with **web_service_call** functions. If you do not specify a service, Service Test creates a script with **soap_request** functions.

To create a server traffic script:

1. Choose **File** > **New** and click **New Single Protocol Script** in the left pane.
2. Select the **Web Services** protocol and click **OK**.
3. Click the **Analyze Traffic** button or choose **Vuser** > **Analyze Traffic**. The wizard opens.

4 Add one or more services to the list. This step is optional.



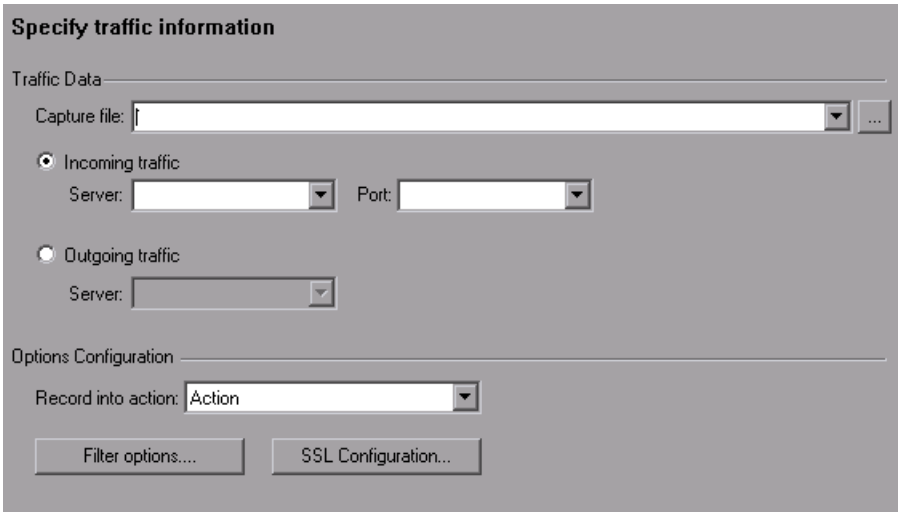
- ➤ To add a new service, click **Import**. In the Import Service dialog box, specify the location of the WSDL. You can specify a URL, File, UDDI server (such as Systinet), or a location in Quality Center. In the Import Service dialog box, you also choose a toolkit for analyzing the service. The selected toolkit will be permanently associated with the script—it cannot be changed. For more information, see "Importing Services" on page 322.
- ➤ To set or view details about the services, click **Details** to open the Service Management window. For more information about Service Management, see Chapter 21, "Managing Web Services."
- ➤ To remove a listed service, select it and click **Delete**

5 On the bottom of the wizard screen, click **Next** to specify the traffic file information. For more information, see below.

6 After providing traffic information, click **Finish** to generate a script.

Specifying Traffic Information

The traffic file contains a dump of all the network traffic. Using the wizard, you specify the location of the traffic file and whether you want your script to emulate incoming or outgoing traffic.



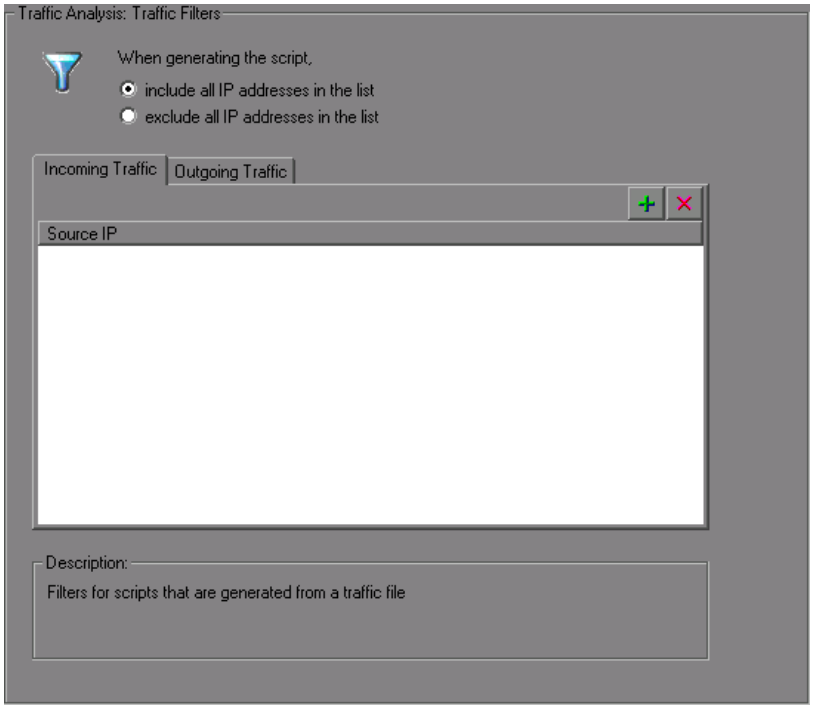
- **Capture file.** The name and path of the traffic file, usually with a cap extension.
- **Incoming traffic:Server/Port.** The IP address and port of the server whose incoming traffic you want to examine.
- **Outgoing traffic:Server.** The IP address of the server whose outgoing traffic you want to examine.
- **Record into action.** The section into which to create the script. If you want to use iterations, specify the **Actions** section.
- **Filter options.** Opens a filter interface allowing you to specify which IP addresses to include or exclude from the script. For more information, see below.
- **SSL Configuration.** Allows you to add SSL certificates to analyze traffic from a secure server with the required credentials. For more information, see "Providing an SSL Certificate" on page 378.

Choosing an Incoming or Outgoing Filter

You can provide a filter to drill down on specific requests going to or from a server, by specifying its IP address and port.

It is also possible to filter your capture file with an external tool before loading it into Service Test. In that case, you may not require additional filtering.

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.



To filter the traffic file:

1 Open the Traffic Filters recording options. Click **Filter Options** in the Specify Traffic Information step, or choose **Tools** > **Recording Options**. Select the **Traffic Analysis:Traffic Filters** node.

2. Select one of the filtering options: **include all IP addresses in the list** or **exclude all IP addresses in the list**.
3. Select the tab that corresponds to your script type: **Incoming Traffic** or **Outgoing Traffic**.
4. Add hosts to the list.

To add a host to the list, click the **Add** button. Specify the IP address of the server you want to add to the list. For incoming traffic, specify the port of the server to include or exclude. Click **OK** to accept the settings.

Click **Delete** to remove an entry.

After the script is created you can change the filters and regenerate the script—there is no need to re-analyze the capture file.

Providing an SSL Certificate

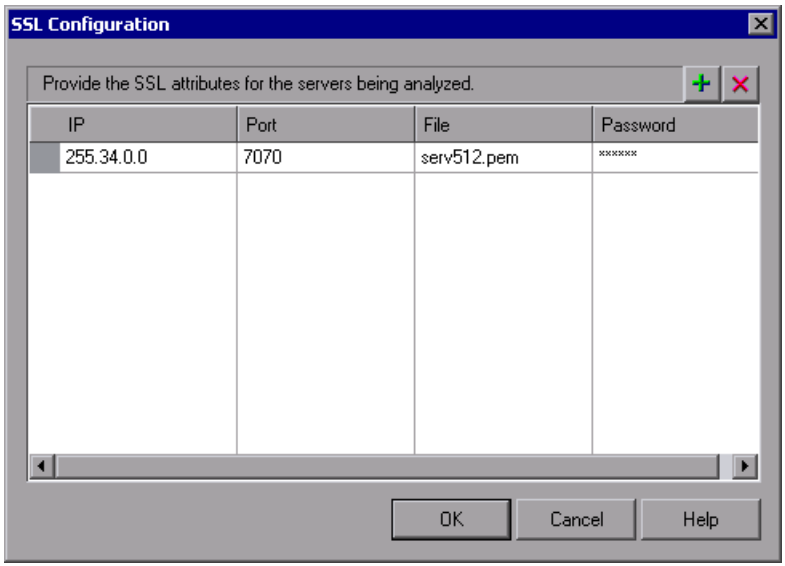
To analyze traffic from a secure server, you must provide a certificate containing the private key of the server.

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.

To specify an SSL certificate:

1. In the Specify Traffic Information screen, click **SSL Configuration**.
2. Add certificates to the list.

To add a certificate to the list, click the **Add** button. Specify the IP address, port, path of the certificate file (with a **pem** extension), and the password for the certificate. Make sure the **pem** file contains the private key. If you are unsure of how to obtain the certificate, contact your system administrator.



Click the **Delete** button to remove an entry from the list.

3 Repeat the above steps for every certificate you want to add.

4 Click **OK** to close the dialog box.

25

Working in the Web Service Call View

You use the Web Service Call view to display snapshots, set properties, and add checkpoints to Web Service calls.

The following information only applies to Web Services and SOA Vuser scripts.

About the Web Service Call View

Using the Web Service Call view, you can view snapshots, set properties, and define checkpoints for each of the Web Service calls in your script.

To open the Web Service Call view, you must be in Tree view. Choose **View** > **Tree View** in the Service Test window to open Tree view.

The Snapshot lets you view the SOAP structure of each step. You can view snapshots of the recording, replay, request, and response. For more information, see below.

The Properties tab lets you configure the settings for each Web Service call. You can set properties in the area of argument values, parameterization, transport layer, and security. For more information, see "Understanding Web Service Call Properties" on page 385.

Checkpoints let you verify your Web service results. You can check for expected values and view the output to see if they were matched. For more information, see "Setting Checkpoints" on page 409.

Viewing Web Services SOAP Snapshots

You can use Service Test's snapshot viewer to examine the SOAP requests and responses that occurred during record and replay. Note that you must replay the session at least once in order to view a replay snapshot.

There are several ways to view the SOAP snapshots:

- Record and/or Replay
- Request or Response Data
- Tree or XML View

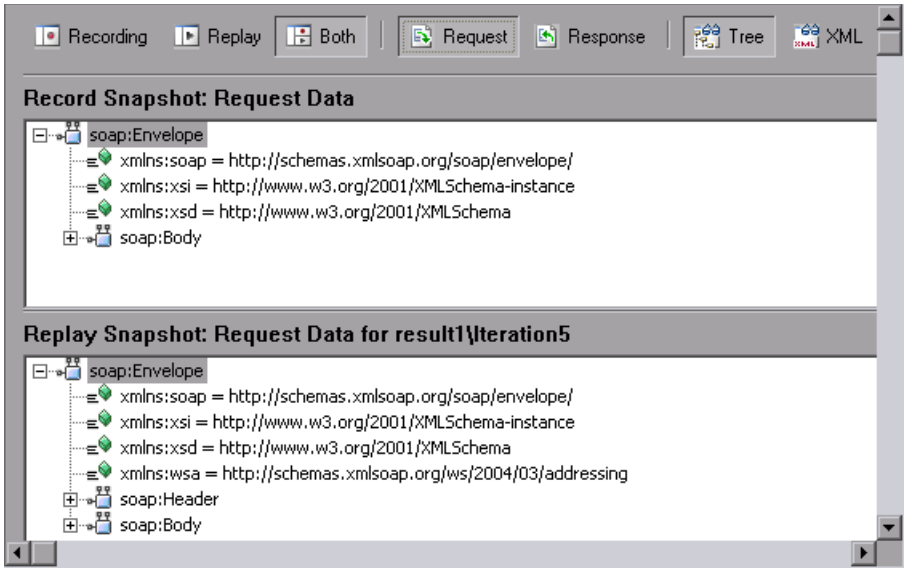
The SOAP Snapshot view also provides you with several utilities that allow you to work with the XML code: **FInd an XML Element** and **Validate XML**.

Using the buttons in the Snapshot window, you can control the view:

In **Tree** view, you can expand the nodes to view the values of the arguments, if they were assigned.

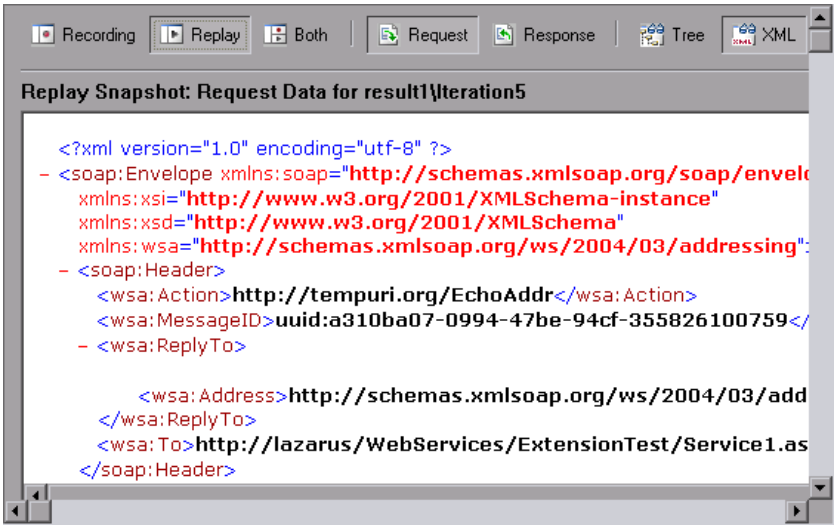
In **Request** view, the displayed values are those that were sent to the server during the Web Services session. In **Response** view, the displayed values are the results returned by the server.

In the following example, the Snapshot window shows the **Record** and **Replay** snapshots of the **Request** data in **Tree** view.



To learn more about a node, select it and choose **Node Properties** from the right-click menu.

In the **XML view**, you can view the whole SOAP message in XML format.



Choosing a Replay Iteration

If you replayed the script with multiple iterations, you can specify which iteration to display in the snapshot. In addition, you can display a snapshot from test results that you saved in a location other than the script's folder.

By default, the Snapshot view shows the last iteration.

To choose an iteration to display:

1. Choose **View** > **Snapshot** > **Choose Iteration**.
2. Select the desired iteration and click **OK**.
3. To display results from another folder, choose **Select Folder** and browse to the location of the test results.

Understanding Web Service Call Properties

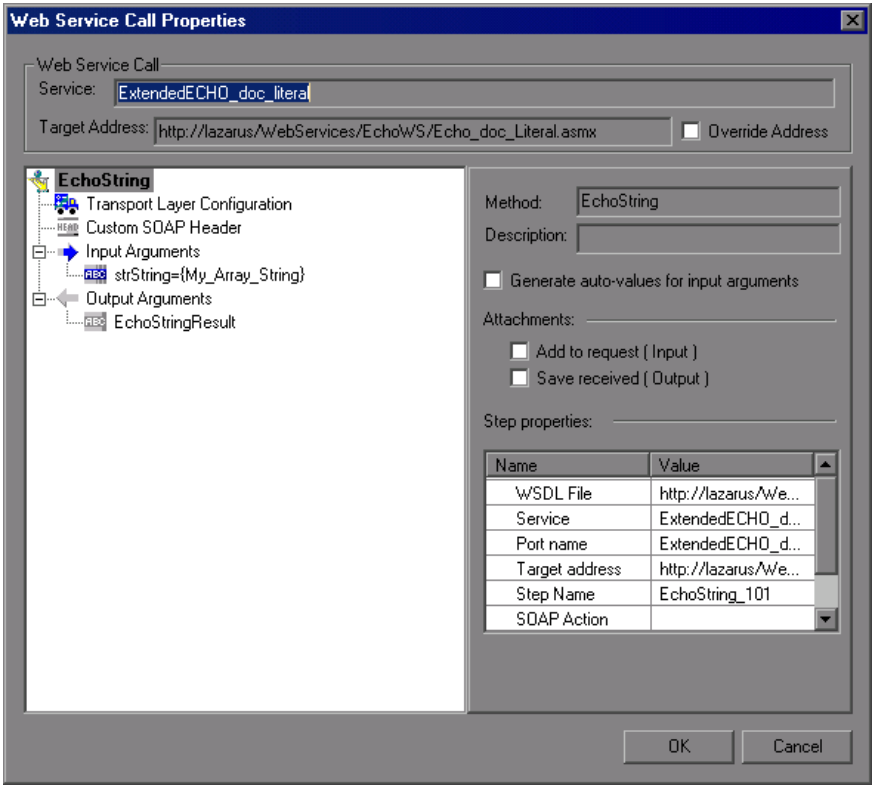
Service Test provides an interface for you to view and modify the properties of each one of the Web Service calls.

Properties describe the behavior of each method within your service. You can set a target address, argument values, parameterization, and transport layer preferences for each of your service's methods.

You can view a step's properties from Tree view (**View** > **Tree view**) in one of the following ways:

- Double-click on a step in the left pane to open the Web Service Call Properties dialog box.
- Select the **Step Properties** tab in the right pane.

The Properties view displays each of the service's operations in a tree hierarchy. The nodes of the tree represent the Transport Layer Configuration, the SOAP header, input arguments, and output arguments.



By default, the script takes the target address from the WSDL. You can override this address for each operation. Select the **Override Address** option and specify the desired **Target Address**.

The contents of the right pane changes, depending on the level of the selected tree node. The following table describes the content for each node:

If you select…	**The right pane shows…**
A method or operation name	➤ The method's properties. ➤ A check box to enable the automatic generation of values for input arguments. ➤ A check box to include attachments.
Transport Layer Configuration	Advanced transport options: ➤ **HTTP/S Transport** with async. or WS-A routing. ➤ **JMS Transport** support with response and request Queue information.
SOAP Header	An edit box to indicate the value of the SOAP header for the current element. For more information, see "SOAP Headers" on page 398.
Input Arguments node	➤ The **Name** of each method and its **Value.** ➤ A check box to enable the automatic generation of values for input arguments.
An individual argument	➤ **Value.** The value of the argument. ➤ **Generate auto-value for this argument**. Insert automatic values for this node.
A complex input argument	➤ **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. To import SOAP, see "Importing SOAP Requests" on page 352. ➤ **Generate auto-value for this argument.** Inserts automatic values for all arguments of this complex type node. ➤ **Add/Delete**. Adds or removes elements from the array.

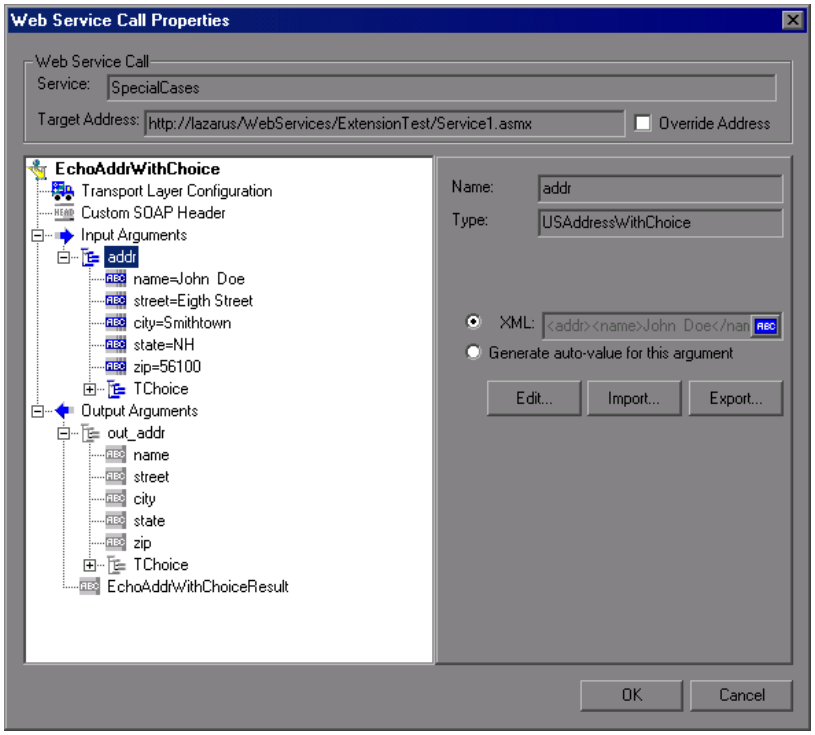
Output Arguments	The **Name** arguments and the **Parameter** that will store the value
Input Attachments	The **Attachment Format** for encoding the soap request: DIME or MIME for the Service Test toolkit, DIME only for .NET, and MIME only for Axis.
Attachment	➤ **Take Data from.** The name of the file to attach or the name of the parameter containing the data. ➤ **Content Type.** The attachment's content type. You can instruct Service Test to detect it automatically or choose a type from the dropdown list or enter a value manually. ➤ **Content ID.** The attachment's ID attribute. You can instruct Service Test to automatically generate a value or specify your own ID. ➤ **Delete Attachment.** A button to remove the attachment.
Output Attachments	➤ **Save All Attachments**. Saves all attachments to parameters: ➤ **Content.** The name of the parameter prefix for storing the attachment. ➤ **Content Type**. The attachments' content type attribute (read-only). ➤ **Content ID**. The attachment's ID attribute (read-only). ➤ **Save Attachments by Index**. Save the attachments by number, beginning with 1. Click **Add** to insert additional attachment indexes.

You can set argument values for the following elements: (manually edited arguments are displayed in blue)

- Input Argument Values
- Output Arguments
- Arrays
- Attachments
- SOAP Headers

Input Argument Values

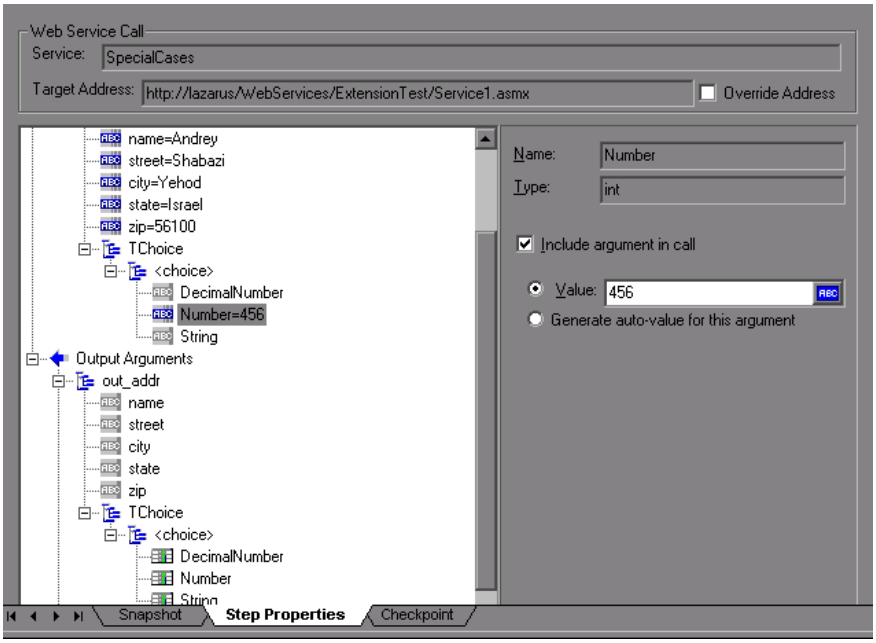
The Input Arguments node lets you define values for all of the input arguments.



- **Value.** You can specify a value for the node manually. To create a parameter for the argument, click the **ABC** icon in the right corner of the **Value** box to open the Select or Create Parameter dialog box.
- **Generate auto-value for this argument.** If you want Service Test to automatically generate a value for this argument, select this option or select the argument in the tree hierarchy and choose **Generate Auto-values** from the right-click menu.

Choice Elements

If your WSDL defines Choice elements, you can view them and set their values in the Properties view.



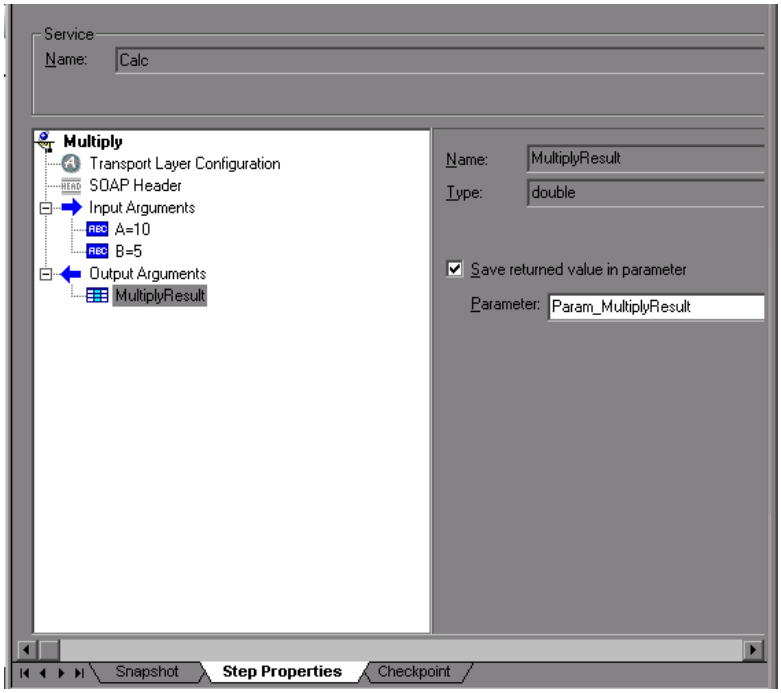
To set a value for a choice element, select the parent element, enable the **Include argument in call** option in the right pane, and provide a value.

To parameterize the argument, click the **ABC** icon. In the Parameter Properties dialog box, provide values for the choice argument. You only need to provide values for one of the choice elements. When running multiple iterations, the script uses the values for the same choice element, according to the assignment method (sequential, unique or random). For example, if your choice elements are **Decimal Number**, **Number**, and **String**, and you provided values for **Number**, the Vuser will always use the **Number** element.

Choice support is provided for both Input and Output arguments, Parameterization, Checkpoints, and Service Emulation.

Output Arguments

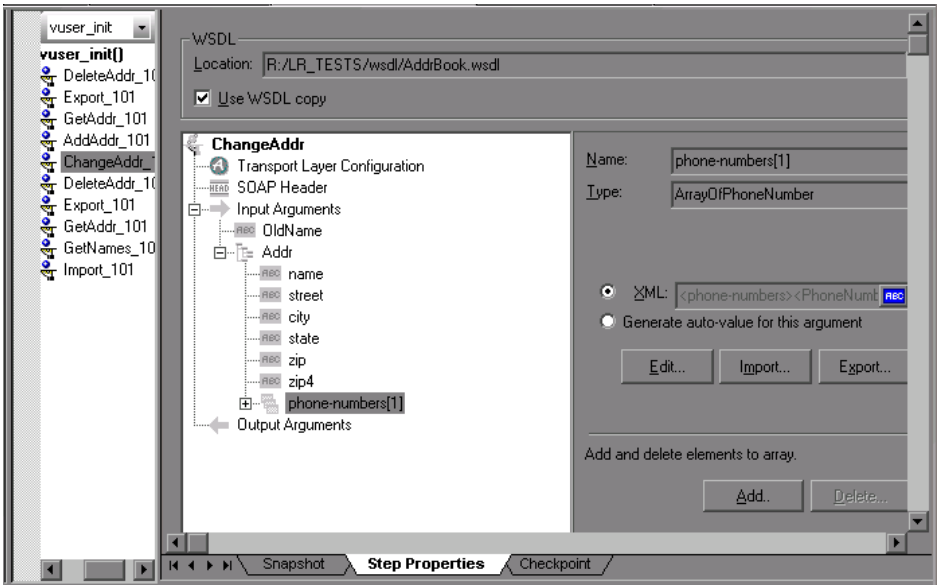
You can view the output argument values and save them to parameters or in an array.



➤ **Save returned value in parameter.** Saves the returned value to a parameter whose name you specify in the text box.

Arrays

To work with an array—for either input or output arguments, select it in the left pane.



➤ **XML.** The path of the XML file containing the values of the array elements. Click the **ABC** icon to replace the XML with an XML type parameter. XML parameterization supports arrays as input arguments. In the XML parameter, you define the number of array elements as required.

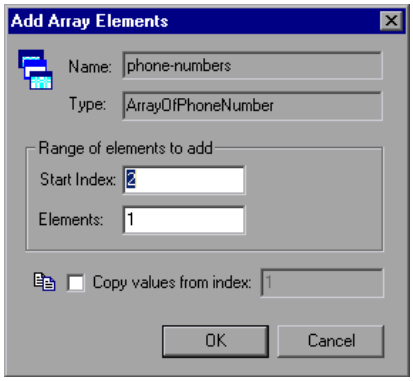
When saving an array to a parameter, the number of array elements per parameter is constant. If you want to run multiple iterations, with each iteration using a different number of array elements, you need to define separate parameters, each containing the desired number of array elements.

For more information about XML parameters, see “Setting Properties for XML Parameters” on page 180.

- **Edit/Import/Export.** To modify complex types and arrays, select the elements and arguments and click **Edit**. Click **Export** to export the selected entry to a separate XML file, or **Import** to load a previously exported XML file. **Note:** This **Import** operation handles XML files that were previously exported—not standard SOAP files. To import SOAP, see "Importing SOAP Requests" on page 352.
- **Add.** Opens the Add Array Elements dialog box, allowing you to add array elements, either simple or complex.
- **Delete.** Deletes array elements. Specify the starting index and the number of elements to remove.

Adding Array Elements

When you click **Add** in the Array Elements section, the Add Array Elements dialog box opens as described below.



- **Start Index.** The index of the first element that will be added.
- **Elements.** The number of elements to add.
- **Copy values from index.** Assigns values of an existing array element to the new elements. Specify the array index of the element whose value you want to use.

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 mechanisms 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.

Service Test supports the sending and receiving of attachments with SOAP messages. You can send Input (Request) or save Output (Response) attachments.

To add or save attachments, select an operation or a method in the left pane to which the attachments will be associated. You can add both input or output attachments

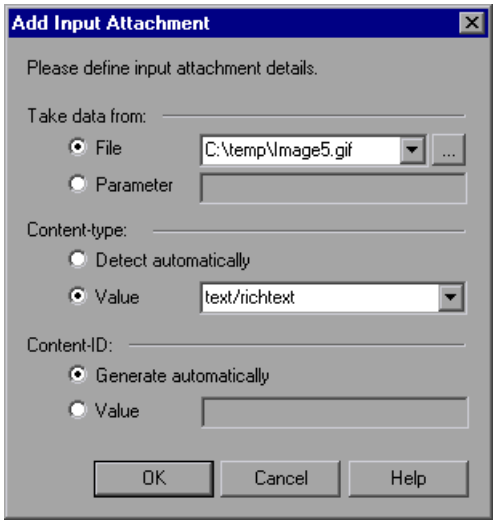
Input Attachments

Input attachments are added to the request message.

To add an attachment to the request:

1. Select the operation in the left pane, to which you want to add the attachment.
2. In the right pane, select **Add to request (Input)**. Service Test prompts you to enter information about the attachment and adds it to the method's tree structure.

The Add Input dialog box opens.



Specify the following information:

- **Take data from.** The location of the data. This can be a file or a parameter that contains the binary data.
  - **File.** You can specify the file location in two ways:
    - 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, Service Test searches for the attachment file in the script's folder. To add it to the script's folder, choose **File** > **Add Files to Script** and specify the file name.
  - **Parameter.** You specify the name of a parameter containing the data.
- **Content-type.** The content type of the file containing the data. The **Detect Automatically** option instructs Service Test to automatically determine the content type. You can also choose from a list of the common content types in the **Value** box, such as text/html, and image/gif or type in another content type.

Content-ID. The ID of the content. By default, Service Test generates this automatically by Service Test and serves as a unique identifier for the attachment. Optionally, you can specify another ID in the **Value** box.

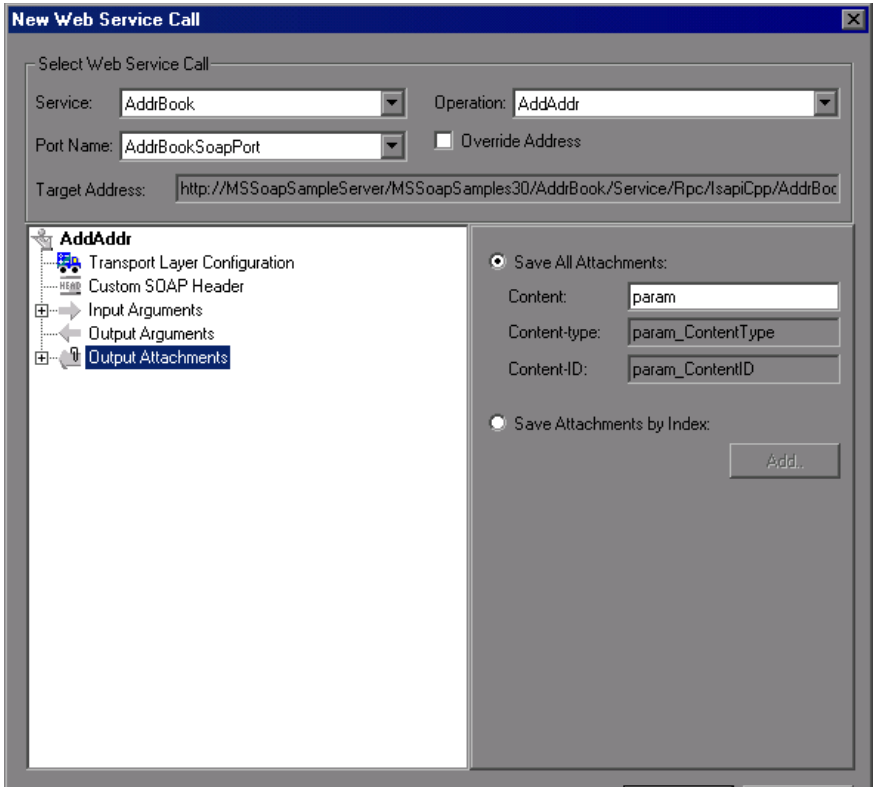
Output Attachments

Output attachments are added to the response message.

To save the response as an attachment:

1. Select the operation in the left pane, for which you want to save the response.
2. In the right pane, select **Save received (Output)**. Service Test adds an Output Attachment node to the method's tree structure in the left pane.

3 Select the desired option: **Save All Attachments** or **Save Attachment by Index** based on their index number—beginning with 1.



When you specify **Save All Attachments,** Service Test 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.

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.

SOAP Headers

This view is available when you select **SOAP header** in the method's tree view. The right pane lets you indicate whether or not to use SOAP headers. To use them, select **Use SOAP header**. Note that you must individually specify SOAP headers for each element. You can import XML code for the SOAP header, or compose your own using the Edit XML option. For more information, see "Editing an XML Tree" on page 401.

Querying an XML Tree

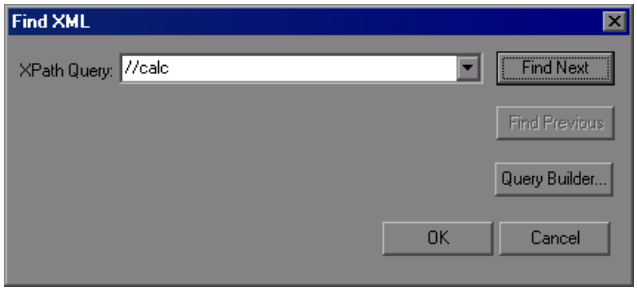
You can query an XML tree in order to locate and examine a specific element and value.

To search an XML, 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.

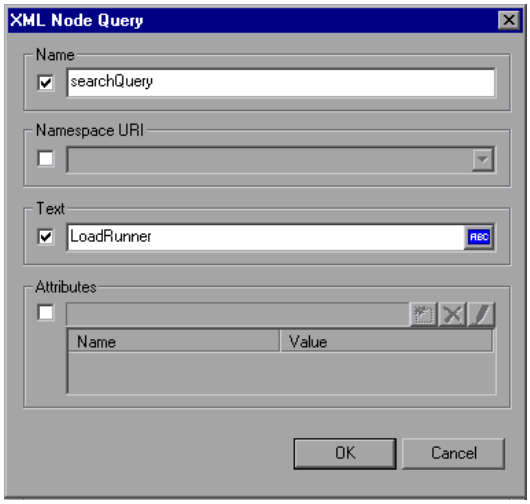
To perform a query:

1. In the snapshot's Tree view, select the Request or Response snapshot and click the **Find XML Element** button .

2 Enter an XPATH query.



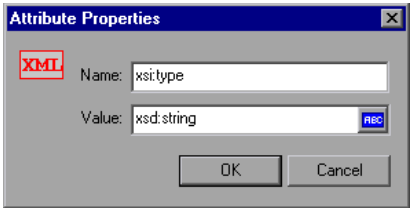
3 Click **Query Builder** for help in building an XPATH query. The XML Node Query dialog box opens. Enable one or more items for searching.



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**.



4 Click **OK** in the XML Node Query dialog box. Service Test places the text of the query in the XPath query box.

5 Click **Find Next** to begin the search.

Working with the XML

Web Services allow you to view and edit your XML.

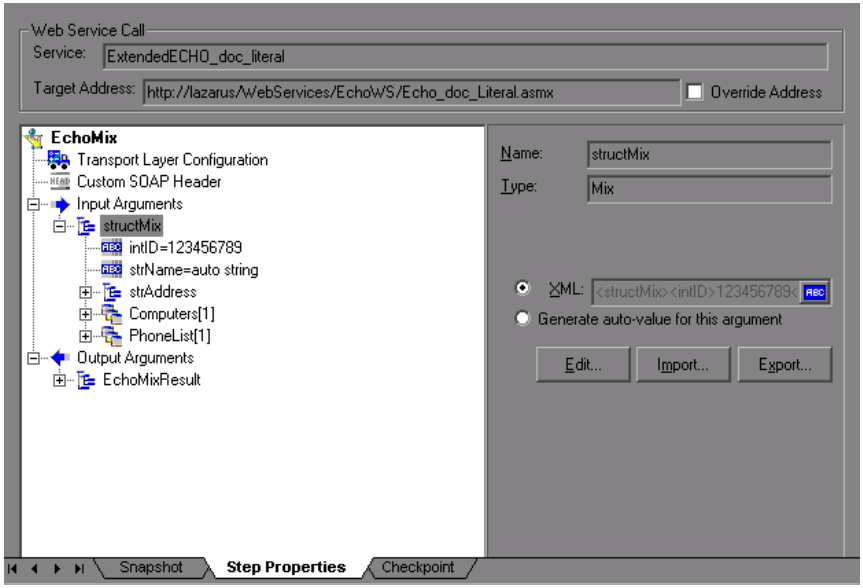
The following sections describe:

➤ Editing an XML Tree

➤ Saving and Copying the SOAP Response

Editing an XML Tree

You can use Service Test's XML Editor to view and edit the XML representation of complex types (structures, objects, etc.) and arrays.

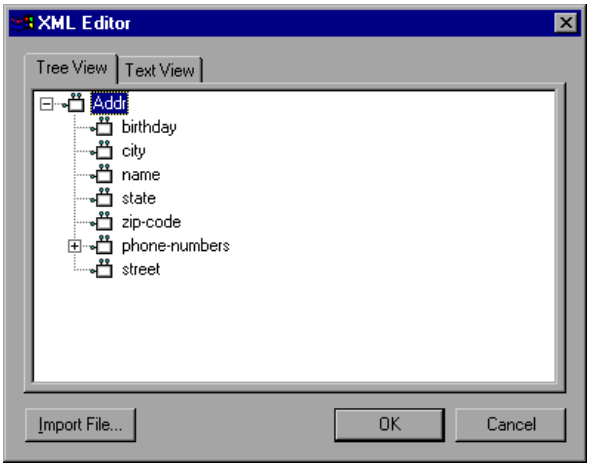


Entering the values for the XML elements is a tedious and error-prone task. Service Test provides you with an interface that simplifies the task of entering, saving, and restoring the information. Once you enter the data manually, you can save it to an XML file using the **Export** option. For subsequent tests, you can import this file without needing to reenter the values a second time.

To edit XML strings:

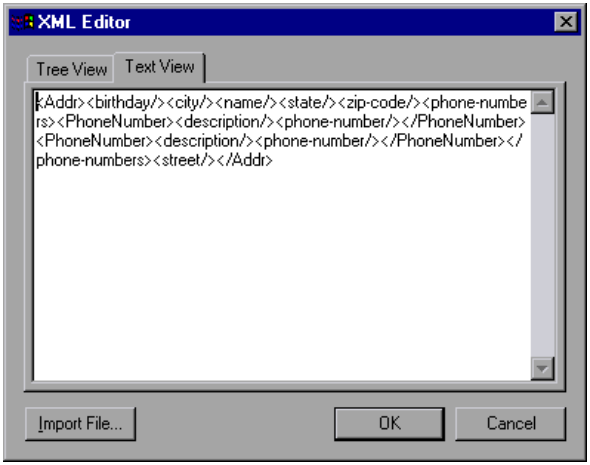
1. Select the Web Service call whose element you want to modify and click the **Step Properties** tab.
2. In the method's tree hierarchy, click on a complex type or array argument. The right-most pane shows the XML code as a single string. Select the **XML** option.
3. To edit the XML code for that entry, click **Edit**. Service Test may issue a warning indicating that only changes to element values and the number of array elements will be saved—not changes in the XML elements themselves.

4 Click **OK**. The XML Editor dialog box opens.



5 In the XML Tree view, double-click on a node to open its property dialog box. Edit the value as required. Click **OK** to save the new values.

6 To edit the code in text mode, click the **Text View** tab. Edit the XML code manually. Click **OK** to save the changes.



7 To import a previously saved XML file, click **Import** and specify the file's location. Edit the file in the XML Editor dialog box.

8 To save your XML data to a file so it can be used for other tests, click **Export** and specify a location.

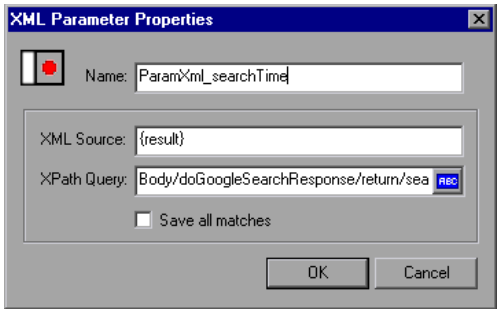
Saving and Copying the SOAP Response

In addition to saving the input argument values as an XML type parameter, you can also save the SOAP response to a parameter or copy it for use within an editor.

To save the SOAP response to a parameter:

1 Switch to the **SOAP Snapshot** tab and select the parent or child element whose value you want to parameterize.

2 Select **Save XML in parameter** from the right-click menu. The XML Parameter Properties dialog box displays the properties of the selected XML element.



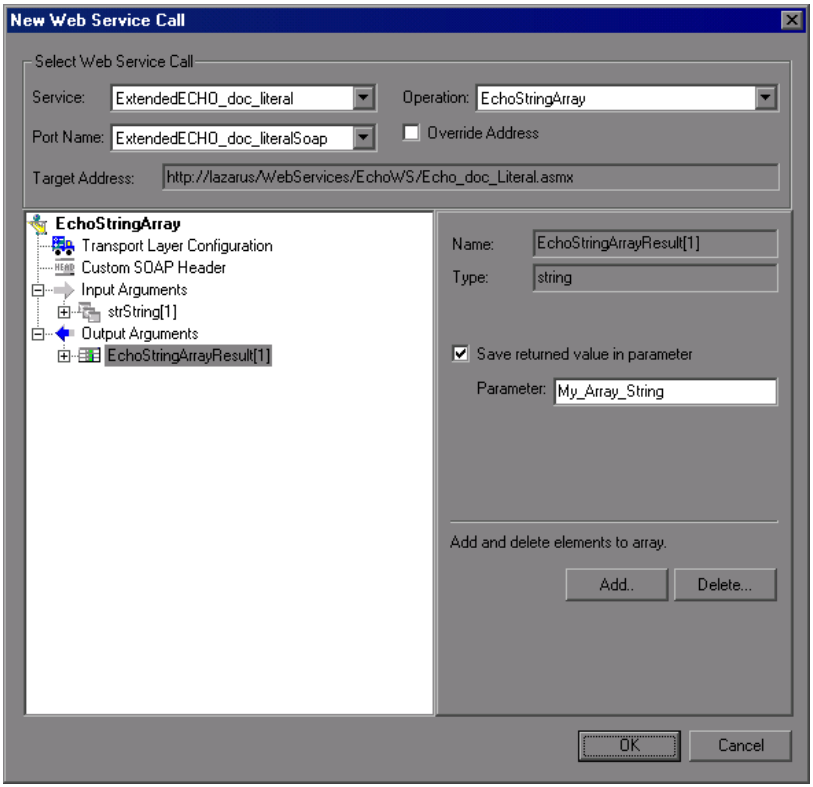
3 Specify a name for the XML parameter, and click **OK**.

To copy the XML structure for use within another editor, choose **Copy XML** from the right-click menu.

Using Web Service Output Parameters

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 an output parameter and reference it at the required point.

In the following example, the output argument is saved to a parameter **My_Array_String**.

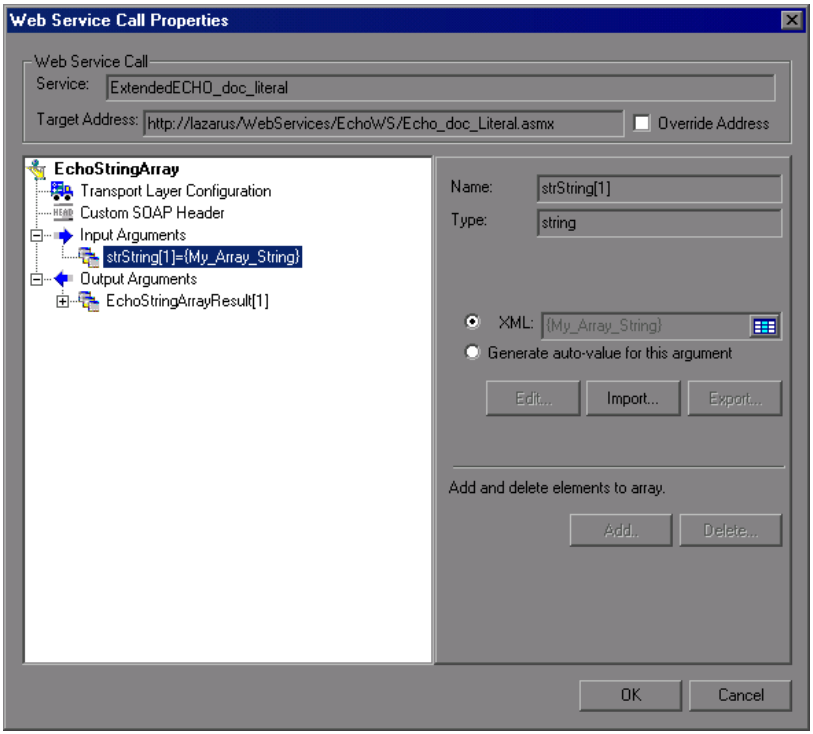


The script shows the saved output parameter as a result argument:

```
web_service_call( "StepName=EchoStringArray_101",
"SOAPMethod=ExtendedECHO_doc_literal.ExtendedECHO_doc_literalSoap.EchoStri
ngArray",
        "ResponseParam=response",
        "Service=ExtendedECHO_doc_literal",
        "Snapshot=t1169994766.inf",
        BEGIN_ARGUMENTS,
        "xml:strString=<strString><string></string></strString>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "EchoStringArrayResult/*[1]=My_Array_String",
        END_RESULT,
        LAST);
```

For information on saving results to parameters, see “Saving Output Parameters” on page 407.

After you save an output parameter, it becomes available for parameter substitution or for other referencing, such as evaluating it and printing its value. In the following example, the saved output parameter, **My_Array_String**, is used as an input argument for a subsequent Web Service call.



For information on using saved output parameters, see “Using Saved Parameters for Input” on page 408.

Saving Output Parameters

You can save multiple result arguments to parameters through the Properties dialog box, or by manually editing them in the script code.

You can also save result parameters from XML actions, and use them as input arguments. For example, if you save the result parameter for **lr_xml_insert**, you can reference the saved parameter in a subsequent Web Service call. For more information, see "Using Result Parameters" on page 494.

To save an output parameter:

1 **View the script in Tree view.**

Make sure you are in Tree view. Otherwise, choose **View** > **Tree view**.

2 **Check for a Service.**

Click the **Manage Services** button to verify that you have imported at least one service. To import a new service, click **Import** in the Service Management dialog box.

3 **View the steps's properties.**

For a new Web Service call, click **Add Service Call**.

For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

4 **Select the output argument.**

In the left pane, select the output argument whose value you want to save to a parameter.

5 **Enable the saving of the output parameter.**

In the right pane, select **Save returned value in parameter**. Accept the default name or specify a custom name.

Using Saved Parameters for Input

After saving output parameters, you can use them in subsequent Web Service calls.

You can also use saved result parameters from XML actions as input. For example, if you saved the result parameter for **lr_xml_insert**, you can reference it in a subsequent Web Service call. For more information, see "Using Result Parameters" on page 494.

To use a saved parameter for input:

1 **View the steps's properties in Tree view.**

For a new Web Service call, click **Add Service Call**.

For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

2 **Select the input argument.**

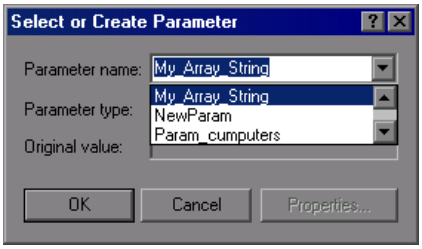
In the left pane, select the input argument whose value you want to replace with a previously saved output parameter.

3 **Open the Select Parameter dialog box.**

In the right pane, select **Value**, and click on the ABC icon adjacent to the **Value** box. The Select or Create Parameter box opens.

4 **Choose an output parameter.**

Select the desired output parameter from the drop-down list and click **OK**.



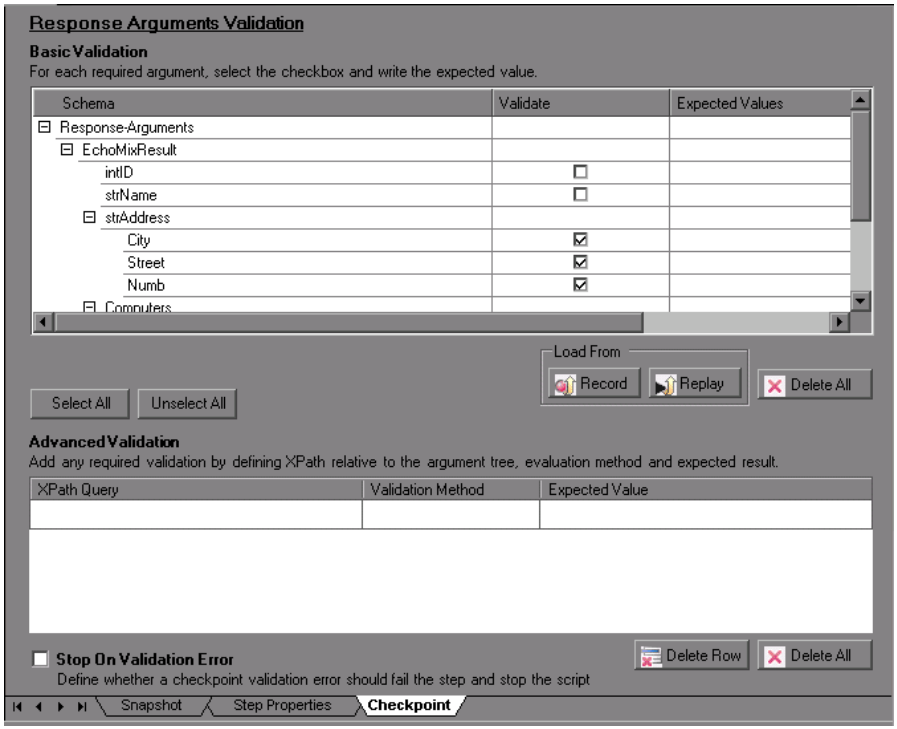
To specify an input parameter in Script view, select the value you want to replace and select **Use Existing Parameters** from the right-click menu. Choose one of the available parameters.

Note: If you modify an output parameter name in Script view, it will not be updated in the parameter list until you switch to Tree view.

Setting Checkpoints

In functional testing, one of the most important tasks is to check the response from the server to confirm that your test performed the actions correctly. In Web Services, the response can contain several arguments, each containing several data items.

Service Test's **Checkpoint** tab is a central point for defining the required checks for your test.



Before running the test, you set expected values for the arguments. You can load a set of expected values as they were captured either during recording or during replay. This is useful when you have many argument values—instead of manually entering values, you automatically load them.

After the test run, you can view the Replay log or the test results and determine if the results were as expected.

Service Test automatically displays all of the method's arguments with a check box. To include a checkpoint in the test, you select its check box. You can load the recorded or replay values (if they exist) and then select only those that you want to check.

An optional **Stop on Validation Error** flag indicates whether or the step fails in case of a checkpoint failure.

In the script, Service Test adds a checkpoint argument for each row that you select in the Checkpoint tab.

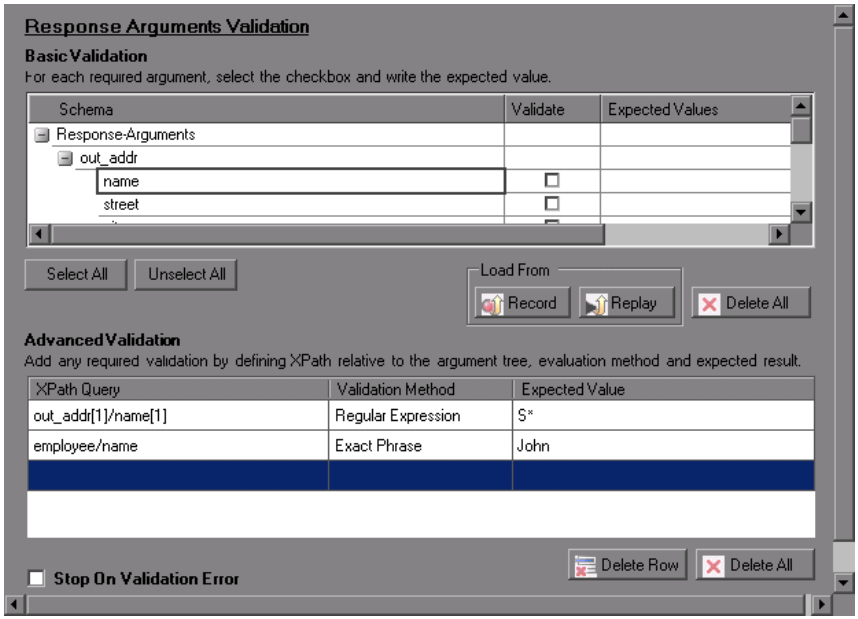
```
web_service_call("StepName=Add_2",
    "SOAPMethod=Calc.CalcSoapPort.Add",
…
    BEGIN_CHECKPOINTS,
        StopOnValidationError=1,
        CHECKPOINT, "XPATH=Result[1]", "Value=13",
        CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
    END_CHECKPOINTS,
    LAST);
```

Service Test also provides support for standard XPATH validations using **lr_xml_find**. For verification of the SOAP body (or SOAP headers with the .NET toolkit), you can use checkpoints. However, when using a toolkit other than .NET, checkpoints are not supported for SOAP headers—instead use **lr_xml_find**. For more information, refer to the *Online Function Reference* (**Help** > **Function Reference**).

Expected Value Types

During replay, Service Test creates a set of expected values for the purpose of validation. These are listed in the upper section, **Basic Validation**.

In addition to the basic validation, you can define **Advanced Validation** to validate a checkpoint on non-leaf nodes or to define expected values in terms of a regular expression.



In Basic validation, Service Test looks for exact matches of the value in the **Expected value** column.

In Advanced validation, Service Test looks for either exact matches or those based on regular expressions.

You define the advanced validation values by entering an XPATH query in the **Advanced Validation** section. To obtain the initial XPATH expression, you can copy it from the basic validation (**Copy Row XPATH** from the right-click menu) and paste it in the **Advanced Validation** section.

Note that when you choose a non-leaf node, you need to supply all of the XML beneath the node.

You can define both basic and advanced validations for the same step.

In the Vuser script, Service Test indicates an exact match by **Value=** and a regular expression with **Expression=**:

```
BEGIN_CHECKPOINTS,
        CHECKPOINT, "XPATH=/AddResult[1]", "Value=50"
        CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
END_CHECKPOINTS,
```

To set a basic checkpoint:

1. In Tree view (**View** > **Tree view**), select a step in the left pane.
2. Select the **Checkpoint** tab.
3. To check for exact matches, specify expected values in the upper **Basic Validation** section:
   - ➤ To manually specify expected values, enter the values in the **Expected Values** column.
   - ➤ To load data from a recording or replay session, click the **Record** or **Replay** buttons in the **Load From** section. Service Test fills in the data as it was captured during record or replay.
4. Select the check boxes in the **Validate** column for all the results you want to check. To select all Basic validation checkpoints, click **Select All**. To clear all of the selections, click **Unselect All**.
5. Click **Delete All** in the upper section to clear all of the expected values.
6. Select **Stop On Validation Error** to instruct the Vusers to fail the step when the replay did not generate the expected values.
7. Run the script and view the Replay log to determine if the service returned the expected values. It is recommended that you enable the advanced log in the run-time settings.

   If there is no match, the Replay log issues an appropriate message:

```
Action.c(14): Failure: checkpoint "/AddResult[1]" expected value="3" actual result="15"
Action.c(14): Error: Web service call "Add" 1/1 checks failed
```

8. Open the Test Results (**View** > **Test Results**) to see a detailed report of the validation. For more information about viewing test results, see below.

To set an advanced checkpoint:

1. In Tree view (**View** > **Tree view**), select a step in the left pane.
2. Select the **Checkpoint** tab.
3. Provide expected values in the bottom section, **Advanced Validation**:
   - An **XPATH Query** expression describing the criteria of the search. You can copy XPATH expressions from the **Basic Validation** section in the upper window. To copy an XPATH expression, select the text in the **Schema** column, and choose **Copy Row XPath** from the right-click menu. In the **Advanced Validation** section, double-click in the next available row and choose **Paste** from the right-click menu. Modify the expression as required.
   - A **Validation Method**: Choose **Exact Phrase** or **Regular Expression** from the dropdown list.
   - The **Expected Value**, either in the form of an exact value or a regular expression.
4. To delete an advanced checkpoint, select it and click **Delete Row** in the **Advanced Validation** section.
5. To delete all of the advanced checkpoints, click the **Delete All** button in the **Advanced Validation** section.
6. Select **Stop On Validation Error** to instruct the Vusers to stop when the replay did not generate the expected values.
7. Run the script and view the Replay log, which contains information on whether or not the match was found.
8. Open the Test Results (**View** > **Test Results**) to see a detailed report of the validation. For more information about viewing test results, see "Viewing Web Services Reports" on page 454.

Parameterizing Checkpoints

You can parameterize the expected values for Checkpoint leaf nodes. This allows the Vuser to use different expected values for multiple iterations.

To replace the expected value with a parameter:

1. Choose **Vuser** > **Parameter List** to open the Parameter List dialog box.

2. Create a new parameter, selecting the appropriate type: **File**, **Table** and so forth. For more information, see "Understanding Parameter Types" on page 149.
3. In Script view, locate the CHECKPOINT section and replace the actual values with the parameter name you created earlier.

```
BEGIN_CHECKPOINTS,
        CHECKPOINT, "XPATH=EchoMixResult[1]", "Value=MyParam"
END_CHECKPOINTS,
```

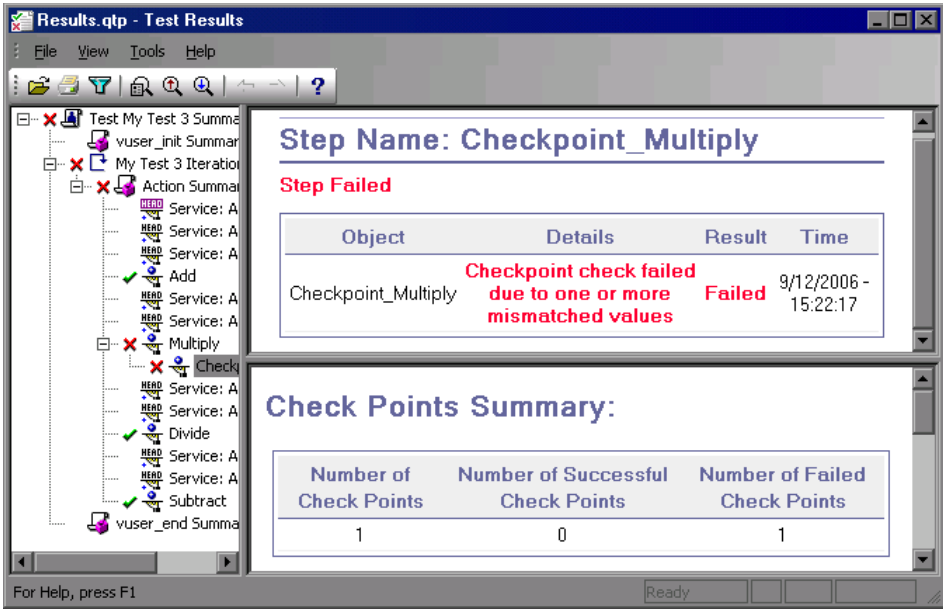
Viewing Checkpoint Results

After running a script, you can view the checkpoint results to see their status—Passed or Failed, and the reason for the failure.

To view the test results for the checkpoints:

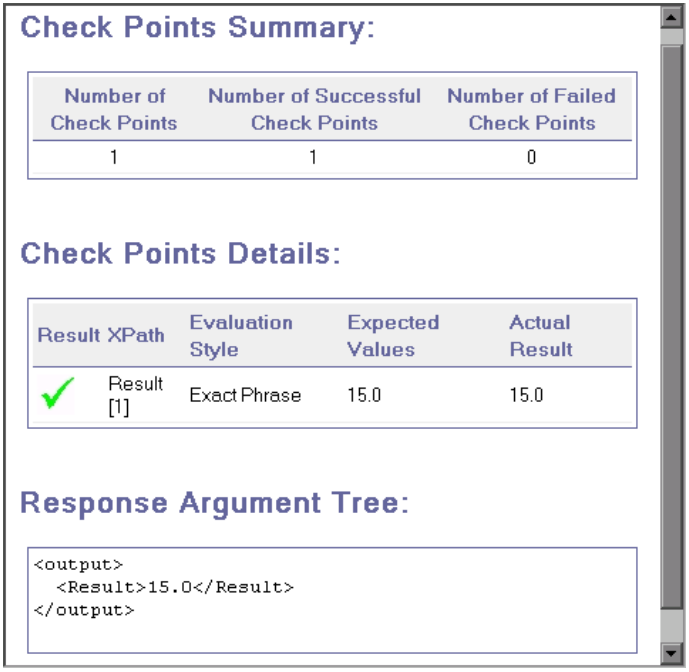
1. Choose **View** > **Test Results** to open the Test Results window.
2. In the left pane, expand the step whose checkpoint you want to view.

3 Click on the Checkpoint step in the left pane. The right pane shows the details about the test run.



The lower pane provides detailed information about the checkpoint:

- The number of successful and failed checkpoints (for multiple iterations)
- The expected values and actual results
- The type of evaluation (exact phrase or regular expression)
- The response argument tree



Check Points Summary:

Number of Check Points	Number of Successful Check Points	Number of Failed Check Points
1	1	0

Check Points Details:

Result XPath	Evaluation Style	Expected Values	Actual Result
✓ Result [1]	Exact Phrase	15.0	15.0

Response Argument Tree:

```
<output>
  <Result>15.0</Result>
</output>
```

For more information about viewing test results, see Chapter 16, “Viewing Test Results.”

26

Setting Advanced Properties for Web Service Scripts

Advanced users can customize Web Service calls by setting the transport layer properties and security policies, and by writing user handlers to define the behavior of the Web Service calls.

The following information only applies to Web Services and SOA Vuser scripts.

About Setting the Transport Layer, Security and User Handlers

You can configure your script with advanced capabilities to customize its behavior.

Using Service Test, you can specify a transport method, such as JMS, for your Web Service calls. You can also set security policies for your Web Service, including standard security with security tokens, SAML security, and JMS.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see "Customizing Web Service Script Behavior" on page 440.

Configuring the Transport Layer

Service Test allows you to configure the transport layer for your services. The transport layer lets you indicate how to transport messages to and from the server—HTTP/HTTPS or JMS (Java Message Service). To learn more about HTTP/HTTPS, see below. For more information about using JMS transport, see "Understanding JMS" on page 427.

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.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls and WS-Addressing.

Asynchronous Messages

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.

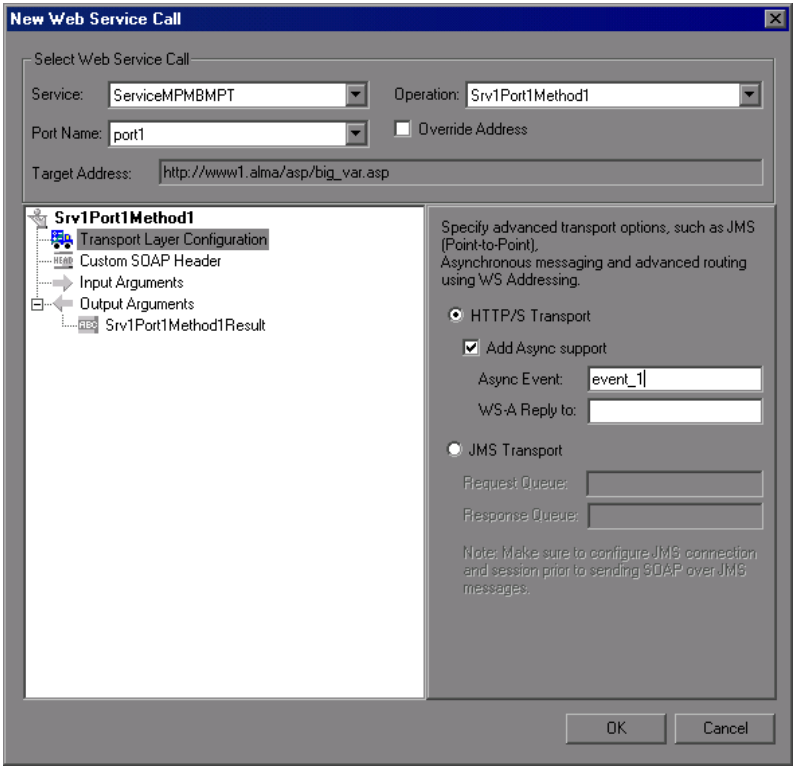
Service Test supports both types of messaging.

In Script view, Service Test 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);
```

To enable asynchronous messaging in Tree view, select the **Transport Layer Configuration** node in the **Step Properties** tab.

Select the **Add Async support** option and specify an event.



When working with asynchronous messaging, you perform synchronization within your script using the **Web Service Wait for Event** step or the **web_service_wait_for_event** function in Script view. This function instructs the Vuser to wait for the response of previous asynchronous service requests. The listener blocks the execution of the service until the server responds.

When adding a Web Service Wait for Event step:

- You list all of the asynchronous events for which you want to wait.
- You then specify whether you want the Vuser to wait for all events to receive a response or just one of them. If you specified **ANY,** then during replay the function returns the name of the first event to receive a response. If you specified **ALL,** any one of the event names is returned.

➤ You provide a timeout in milliseconds. If no events receive responses in the specified timeout, **web_service_wait_for_event** returns a NULL.

In the following example, the web_service_wait_for_event call waits forty milliseconds for any of the events: event_1, event_2, or event_3.

```
web_service_wait_for_event("StepName=First_Wait",
        "Quantifier=ANY",
        "Timeout=40",
        BEGIN_EVENTS,
        "event_1",
        "event_2",
        "event_3",
        END_EVENTS,
        LAST);
```

When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

For additional information about the **web_service_wait_for_event** function, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

You can also indicate the location you want the service to reply to when it detects an event, using WS-Addressing. For more information, see the section below.

WS-Addressing

WS-Addressing is a specification that defines a standard for allowing Web Services to communicate addressing information. It identifies 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.

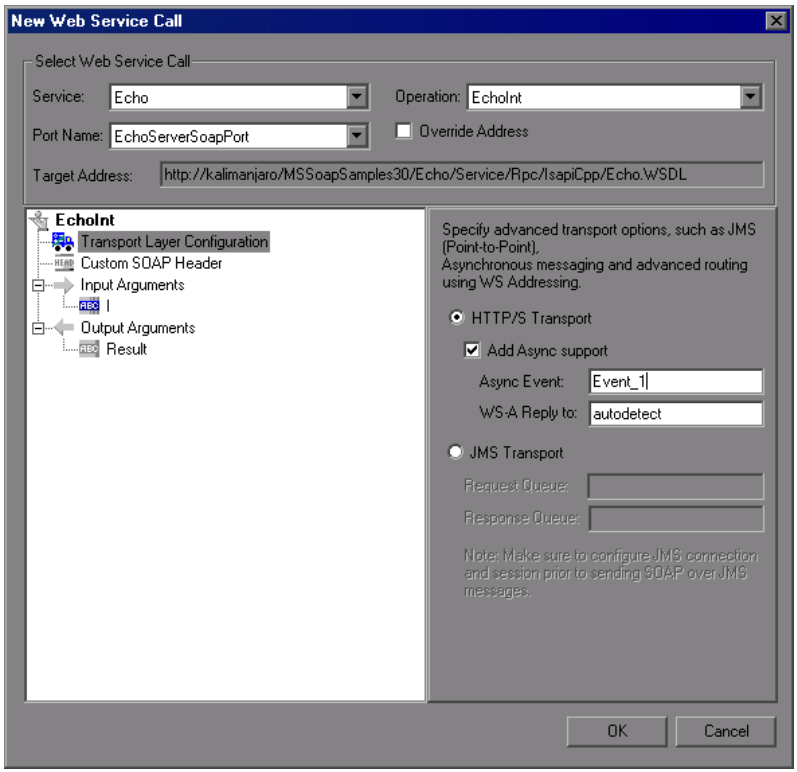
The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

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 by Service Test in the background.

```
<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>
```

To create a SOAP request using WS-Addressing in Service Test, you specify a **WSAReplyTo** entry in the Transport Layer Configuration's node under the **Step Properties** tab.



For the **WSAReplyTo** argument, you can specify an IP address or **autotdetect** to instruct the service to detect the host name of the machine. This is useful when running the same script on several different machines.

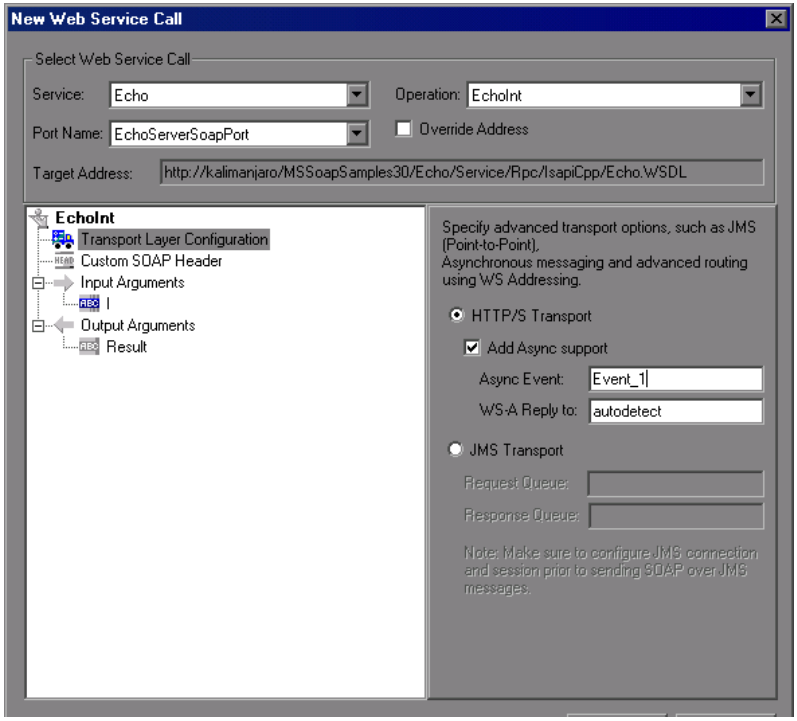
In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
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);
```

WS-Addressing calling may be issued in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, you leave the **Async Event** box empty in the Transport Layer options. In Script view, you remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

To add support for asynchronous messages and WS-Addressing:

1 For a new Web Service call, select the **Transport Layer Configuration** node. For an existing Web Service call, select the step in Tree view, and select the **Step Properties** tab. Select the **Transport Layer Configuration** node.

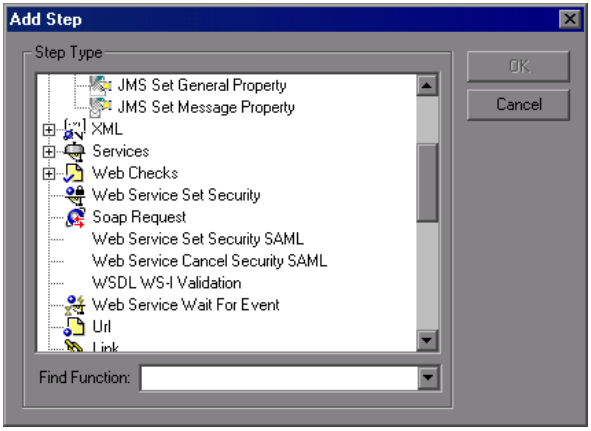


2 To mark the Web Service call as an asynchronous message:

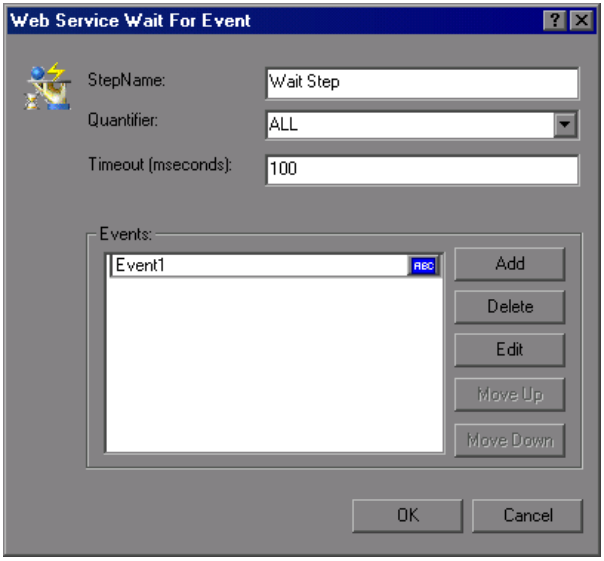
- ➤ Select the **Add Async support** option. This is only enabled for HTTP/S type transport.
- ➤ Provide an event name in the **Async Event** box. This can be an arbitrary name.

3 In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. The server will reply to the specified location when the event occurs.

4 For asynchronous events, choose **Insert** > **New Step** and add a **Web Service Wait For Event** step after the Web Service call step.



5 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.



6 Use the Edit, Move Up, and Move Down buttons to manipulate the events.

7 Click **OK**.

Understanding JMS

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. Service Test supports the sending and receiving JMS messages to a queue from a Web Service call and by using JMS script functions.

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, Service Test lets you set a timeout for received messages and the number of JMS connection per process.

You can configure all of these settings through the JMS run-time settings, as described in "Setting Web Services JMS Run-Time Settings" on page 452.

JMS Script Functions

Service Test uses its JMS API functions to implement the JMS transport. Each function begins with a **jms** prefix. For example, **jms_receive_message_queue** Receives a message from a queue.

For additional information about the JMS functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

Using JMS as a Transport Layer

The JMS transport for Web services allows users to send SOAP messages using a JMS transport instead of the common HTTP transport.

A Web Service can:

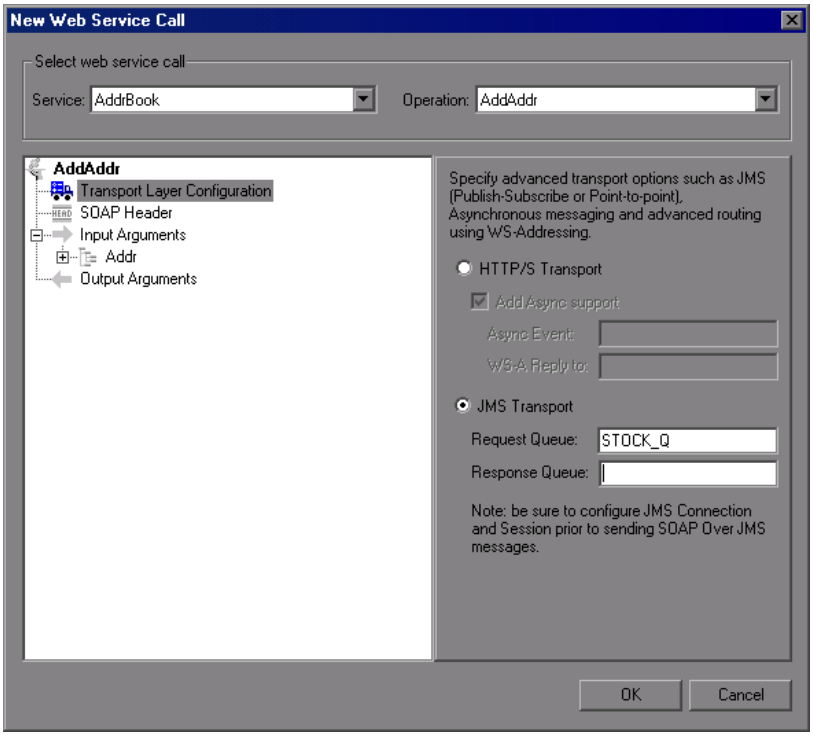
- Import a WSDL and then specify JMS instead of HTTP as a transport layer.

- Record SOAP messages using HTTP, and use the recorded script for sending the messages through JMS transport.
- Send and/or receive general JMS messages to a JMS destination (queue) from the Web Services script.

The basic steps in transmitting messages over JMS protocol are:

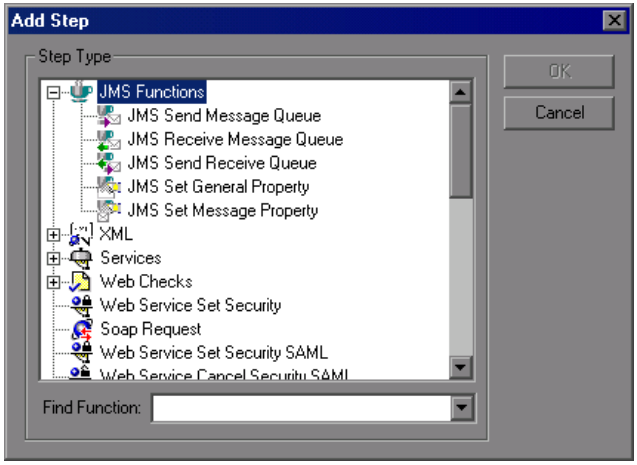
1. Import a WSDL file. For more information, see “Importing Services” on page 322.
2. Setting the queue information, as described below.
3. Configuring the JMS run-time settings before running the script, as described in “Setting Web Services JMS Run-Time Settings” on page 452.

   For each Web Service call step, you can specify JMS Transport and the Request and Response queues.



To specify JMS transport information:

1. In Tree view, select the **Step Properties** tab and select the **Transport Layer Configuration** node.
2. Select the **JMS Transport** option.
3. Specify the names of the **Request Queue** and **Response Queue**. You can use the same queue for the request and response.
4. To manually set additional transport information or get and set properties, add one of the JMS functions. Choose **Insert** > **New Step** and expand the **JMS Functions** node.



The current solution is a replay only solution and does not allow recording JMS messages sent between the client and server. Service Test only supports synchronized calls.

For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

JMS Message Types

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

Service Test attempts to resolve the desired format based on the message's content type. If the content type is **text/***, it sends the message in TextMessage format. If the content type is not **text/***, it sends the message in BytesMessage format.

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 *HP LoadRunner Online Function Reference*.

Creating Web Service Security Policies

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 that is limited to point-to-point communication.

To allow you to send your messages securely, Service Test supports several security mechanisms: Security Tokens and SAML.

Note: If your WSDL is located in a secure location, you must provide the security information through the Service Management dialog box. For more information, see "Specifying WSDL Connection Settings" on page 327.

For more information on tokens, see below.

For more information on SAML, see "Setting SAML Options" on page 437.

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, Service Test 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 **Logical Name**, **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.

  Service Test supports tokens based on both Kerberos and Kerberos2 security tokens. The primary difference between the Kereberos 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 **Logical Name** 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 **Logical Name**, **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 **Logical Name** 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 more information about configuring tokens, refer to the *Online Function Reference* (**Help** > **Function Reference**).

When you add a Web Services Set Security step to your script, Service Test adds a **web_service_set_security** function that contains arguments with the tokens, message signatures, and encryption that you defined in the security properties.

```
web_service_set_security(
   SECURITY_TOKEN, “Type=USERNAME”, “TokenName=mytoekn1”,
   “UserName=bob”, “Password=123”, “PasswordOptions=SendNone”,    “Add=True”,
LAST);
```

Note that 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 does offer 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 passwords.

  Service Test allows you to supply information about the encryption and message signatures.

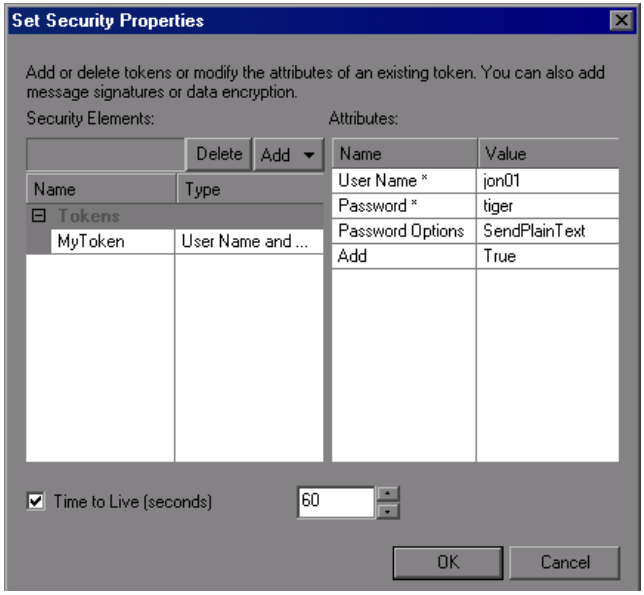
Note that parameterization is not supported for message signatures and encryption arguments. For more information on adding message signatures and encryption to your script, see below.

Setting Web Services Security

While developing a script, you can add security to your Web Service calls using standard WS-Security.

To add Web Service security:

1. Click at the appropriate location in your script. To apply the security to the entire script, place the cursor at the beginning of the script.
2. Choose **Insert** > **New Step** to open the Add Step dialog box.
3. Select **Web Services Set Security** and click **OK**. The Set Security Properties box opens.



4 Click **Add** to add a new token. The Add Token dialog box opens.

5 Choose a token type. For information about the token types, see "Security Tokens and Encryption" on page 431.

In the **Logical Name** box, assign an arbitrary name for the token to be used by Service Test 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, choose **True**. To exclude the token from the SOAP envelope header, choose **False**.

6 To specify a time for which the message packet is considered valid, select **Time To Live** and specify the time in seconds.

7 Click **OK**. Service Test inserts a Web Services Set Security step at the location of the cursor.

Setting SAML Options

Service Test 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. To set SAML security, add a **Web Services Set Security SAML** step. To remove the security, insert a **Web Services Cancel Security SAML** step.

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** function.

Policy Files

SAML policy files follow the WSE 3.0 standard and define the attribute values for the SAML security. By default, Service Test 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.

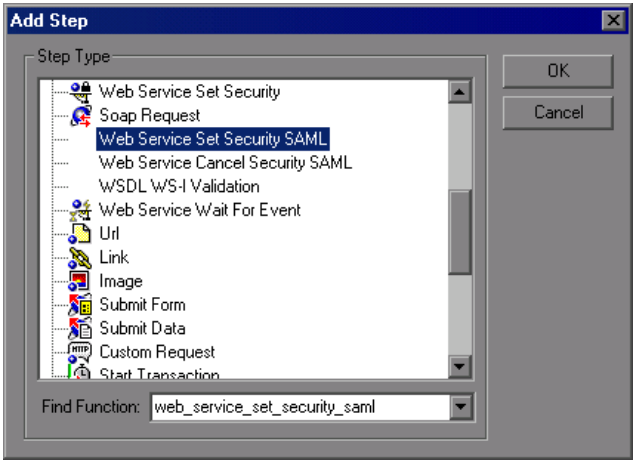
You can modify the policy file to include values for the security parameters, such as username and certificate information. When adding a SAML security step to your script, if you explicitly specify values for the security arguments, they override the values in the policy file.

If you make changes to the default policy file, it is recommended 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, refer to 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.

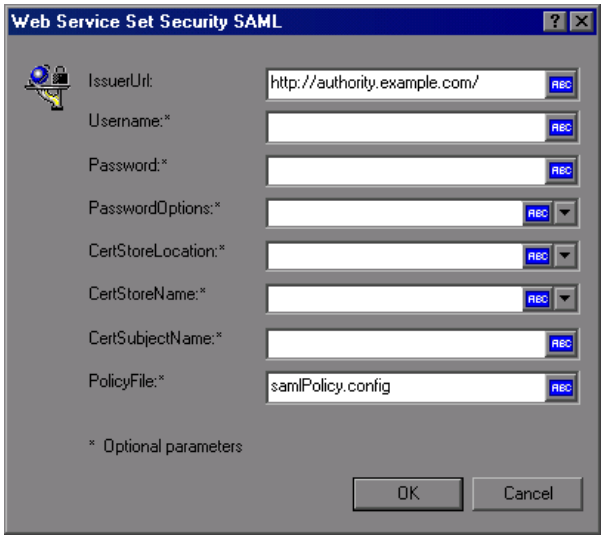
To add SAML security:

1. Click at the appropriate location in your script. To apply the security to the entire script, place the cursor at the beginning of the script.
2. Choose **Insert** > **New Step** to open the Add Step dialog box.



3. To add SAML security, choose **Web Services Set Security SAML**.

Enter the desired information. If you enter values into this dialog box, they override any values that were set in the policy file. You must provide an Issuer URL, also known as the **STS URL**.



To use a different policy file, specify it in the **Policy File** box. Specify a full path, or a file location relative to the script's path.

4 To remove the security, choose **Web Services Cancel Security SAML.** The security is cancelled from that point onward.

For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

Customizing Web Service Script Behavior

Service Test provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are:

- User Handlers
- .NET Filters
- Configuration Files

User Handlers

Service Test allows you to call user handlers to process SOAP requests and responses. Using the handlers, you can retrieve and modify the SOAP envelope, get or set parameters, and issue log messages. You can also use the handler mechanism to add security features, message compression, and filters.

You can implement a user handler in several ways:

- Defining a Handler Function in a Script
- Creating a Custom User Handler as a DLL

Defining a Handler Function in a Script

For basic implementation of a user handler, you define a user handler function within your Vuser script:

int MyScriptFunction(const char* pArgs, int isRequest)

```
int MyScriptFunction(const char* pArgs, int isRequest)
{
…
}
```

The **pArgs** argument contains the string that is specified in **UserHandlerArgs** argument of **web_service_call** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

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, specify the function name as a value for the **UserHandlerFunction** argument in the relevant Web Service Call step.

```
web_service_call(
…
"UserHandlerFunction=MyScriptFunction",
"UserHandlerArgs=<handler arguments>",
  LAST);
```

Service Test 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;
```

Overriding the Transport Layer

You can write a user handler function to override the transport layer. In this case, Service Test will 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, you can set the response envelope with the command:

```
lr_save_string(someResponseEnvelopeStr, "SoapEnvelopeParam");
```

To apply an alternate transport layer, specify **ReplaceTransport** as an value for the **UserHandlerOrder** argument, and define the transport layer in the handler function.

```
web_service_call(
…
"UserHandlerFunction=<Transport HandlerFunction>",
"UserHandlerArgs=<handler arguments>",
"UserHandlerOrder=ReplaceTransport"
…
  LAST);
```

Creating a Custom User Handler as a DLL

You can also define a user handler by creating a DLL file through Visual Studio and the handler API. The API header file, **LrWsHandlerAPI.h**, located in the **ServiceTest/include** folder, contains many in-line comments and descriptions.

Service Test provides a sample Visual Studio project that can be used as a template for creating a handler. The sample retrieves the request and response envelope and saves it to a parameter. This sample is located in the **ServiceTest/samples/WebServices/SampleWsHandler** folder. 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 **ServiceTest/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.

Configuring the User Handler

You can declare the user handler DLL globally or locally.

To use the handler globally, for all requests in the script, add the following section to the **default.cfg** file located 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>
UserHandlerOrder=<BeforeSecurity/AfterSecurity/AfterAttachments/Replace
                  Transport>
```

Note: If you copy the script to another machine, it retains the handler information, since it is defined in script's folder. A user handler defined locally for a specific step in the script, overrides the global handler settings (defined in the script's **default.cfg** file).

Note: The user handler DLL should be accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the **ServiceTest/bin** folder.

Implementing the User Handler

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 **ServiceTest/include** folder.

.NET Filters

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.

You can apply a .NET filter to your messages using the user handler mechanism.

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
```

Configuration Files

The **mmdrv.exe.config** file, located in the **Service Test/bin** folder, is a standard .NET configuration file, and contains information such as the WSE configuration. Use the filter with the **Input** prefix if your filter is derived from SOAP input, or the **Output** prefix if your filter is derived from SOAP output.

If your application has its own configuration file, **app.config**, you can implement it in several ways:

- Save it as **mmdrv.exe.config**, overwriting the existing configuration file. This will apply your configuration information to all scripts on the machine.
- Save **app.config** to the script's folder. The settings in the **app.config** file override the ones in **mmdrv.exe.config**. In addition, if you save it to the script's file, it will always be associated with the script, not requiring you to copy it over separately to other machines.

In addition, the configuration file contains security information. You can configure whether or not to allow unsigned test certificates.

By default, Service Test allows unsigned certificates to facilitate testing. To disallow unsigned certificates, modify the **allowTestRoot** flag in the **mmdrv.exe.config** file to false.

```
<security>
  <x509 storeLocation=”currentuser” alllowTestRoot=”false”
```

27

Running SOA/Web Services Scripts

After you create an SOA/ Web Services scripts, you run it to make sure it is functional. After you run the script, you can view the test results to see whether the services performed as expected.

The following information only applies to Web Services and SOA Vuser scripts.

About Running Web Services Vusers

In Tree view, Service Test provides three tabs that allow you to understand and examine your script:

- **Snapshot.** You can use Service Test's snapshot viewer to examine the SOAP requests and responses that occurred during record and replay. For more information, see "Viewing Web Services SOAP Snapshots" on page 382.
- **Step Properties.** Provides details of each step in your script, along with its argument values, attachments, SOAP headers, and transport layer configuration. For more information, see Chapter 25, "Working in the Web Service Call View."

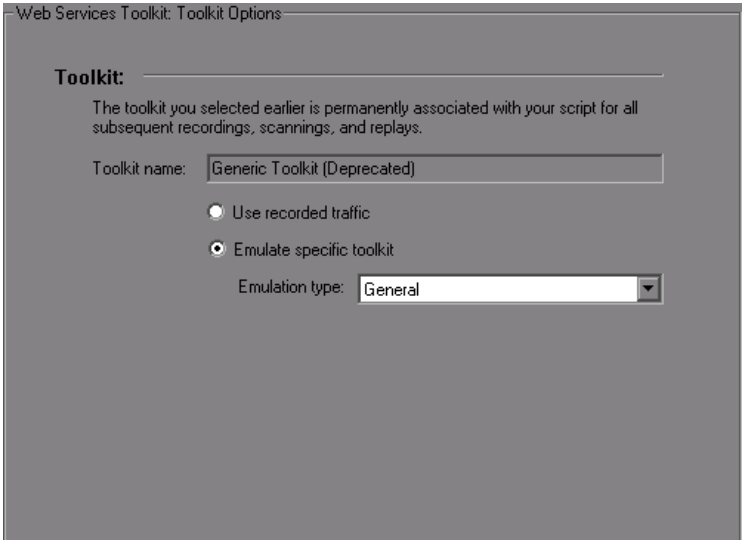
➤ **Checkpoint.** Verification points that help your determine whether or not your service produced correct results. For more information, see "Setting Checkpoints" on page 409.

Before running the script, you can set run-time settings that help you emulate real users more accurately. These settings include general run-time settings (iteration, log, think time, and general information), and Web-Services related settings (Toolkit and JMS).

For details, see the following sections or Chapter 13, "Configuring Run-Time Settings."

Setting Web Service Toolkit Run-Time Settings

You can configure the run-time settings to specify a toolkit emulation for scripts created with the Generic Solution toolkit. These settings do not apply to the .NET and Axis/Java toolkits.



You can use the recorded traffic or specify an emulation type:

Use Recorded Traffic. For scripts that you created by recording a client application, this option lets you emulate the client using the styles and attributes in your recorded script.

Emulate specific toolkit. Emulates a specific toolkit, and not the attributes from the recording. You can specify the type of emulation—MS SOAP, Glue or a general one.

To set the Web Services run-time settings:

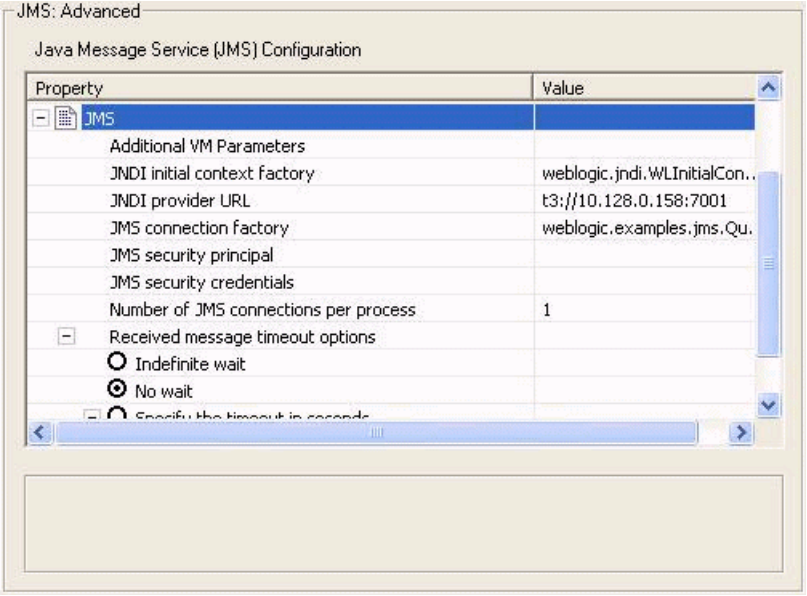
1. Open the Run-Time settings dialog box (**Vuser** > **Run-Time Settings** or F4) and select the **Web Services:Toolkit Options** node.
2. Indicate the desired option: **Use recorded traffic** or **Emulate specific toolkit**.
3. If you chose to emulate a specific toolkit, select an **Emulation type**.
4. Click **OK**.

Setting Web Services JMS Run-Time Settings

To use JMS as a transport for Web Service calls, there are several resources that need to be allocated and configured. Those resources include the JVM, JNDI initialization parameters, JMS resources, and timeout values.

Service Test lets you configure some of those resources through the the run-time settings.

You can set options in the area of VM (Virtual Machine), the JMS connections, and message timeouts.



VM

- **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 Service Test.
- **JVM Home.** The location of the external JVM. This should point to the JDK home directory, defined by JDK_HOME. Service Test supports JDK 1.4 and above.

- **Classpath.** The vendor implementation of JMS classes together with any other required supporting classes, as determined by the JMS implementation vendor

JMS

- **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. Choose a context factory from the list or provide your own.
- **JNDI provider.** The URL string of the service provider. For example:
  Weblogic - t3://myserver:myport
  Websphere - iiop://myserver:myport
- **JMS connection factory.** The 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 less connections you have per process, the better your performance.
- **Receive message timeout options.** The timeout for received messages. The default is **No wait**.
  - **Indefinite 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. (default)
  - **Specify the timeout in seconds.** Manually specify a timeout value for the message. If the timeout expired and no message arrived, the operation fails.

User defined timeout. Specify the amount of seconds to wait for the message before timing out. The default is five seconds.

Using Web Services Functions

Web Service functions call services and provide security and synchronization. The Web Service functions begin with the web_service prefix. For example, **web_service_call** performs a SOAP request from a WSDL and its argument data.

To perform a SOAP request from raw data, use the **soap_request** function.

In addition, you can enhance your script with JMS functions, **jms_<suffix>** or XML functions, **lr_xml_<suffix>**. For more information, refer to the *Online Function Reference* (**Help** > **Function Reference**).

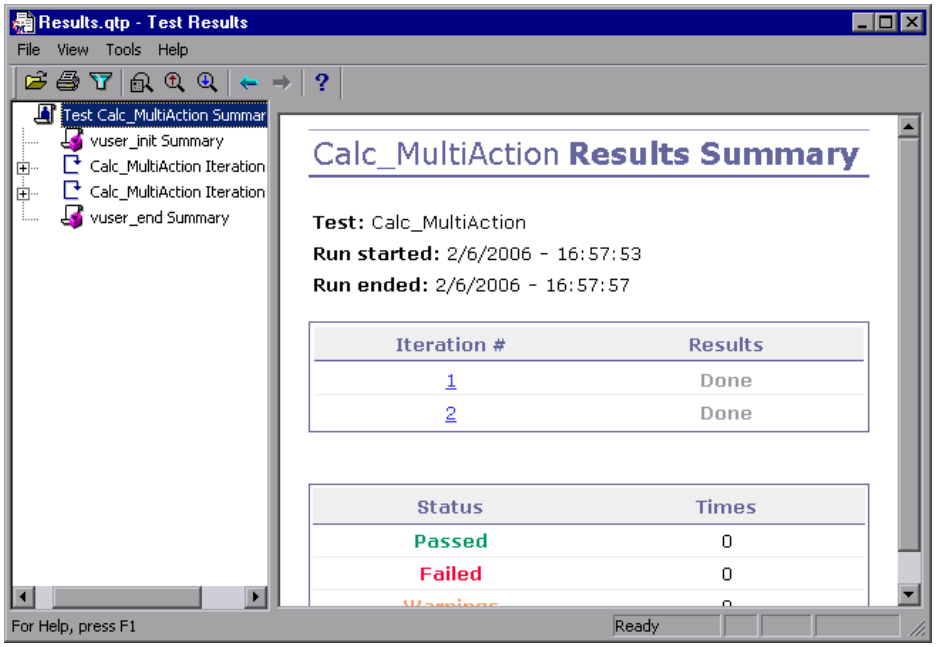
Viewing Web Services Reports

After you run a Web Services script, you can view a summary of the test results using the Test Results viewer. The viewer also shows the results of the checkpoints.

This section describes the Summary report's Web Services information. For general information about the Test Results utility and the available views, see Chapter 16, "Viewing Test Results."

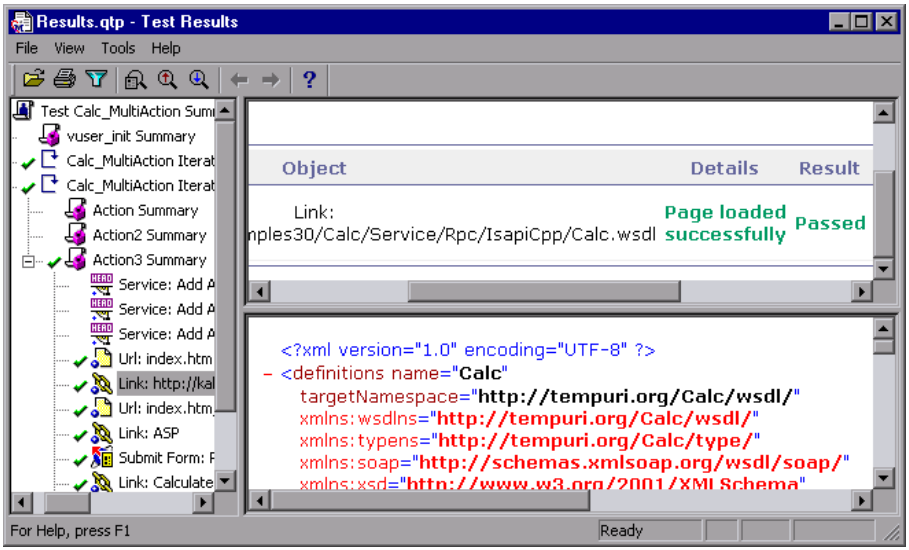
To open the Summary report, choose **View** > **Test Results**.

The test results are divided into iterations, actions, and steps.



The Results report marks a successful step with a green check mark and a failed step with a red **X**. An iteration is only marked as successful if all of its steps and actions have succeeded.

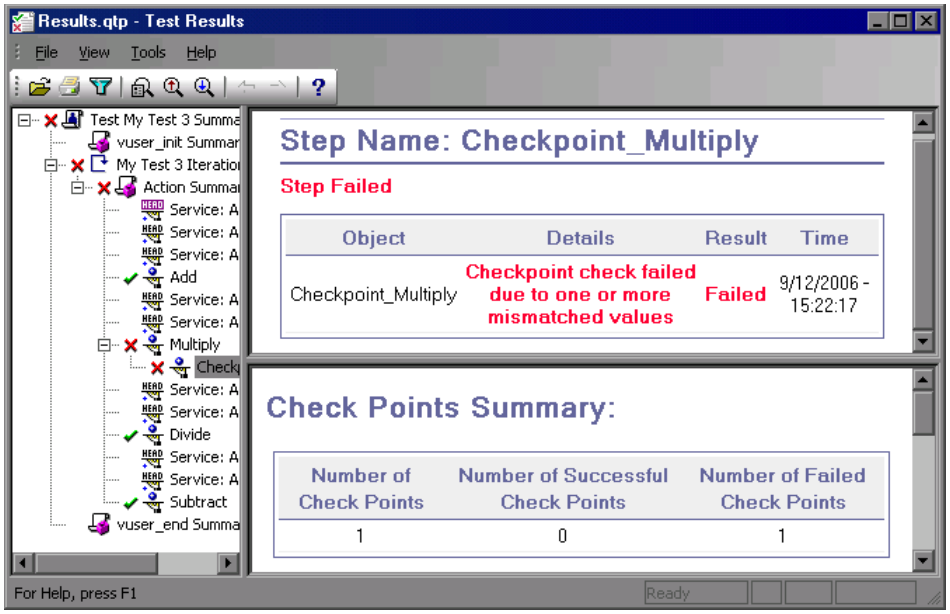
For Web Service calls, the bottom pane of the Results window displays the contents of the SOAP response.



If Service Test cannot interpret the script or if it encounters another type of error, the report displays a message in the right pane stating the problem.

The Results window also shows checkpoint results. It provides a summary with a reason for the failure. It also provides the Expected Value and Actual Results.

To view the checkpoint details, expand the appropriate step in the left pane and click the **Checkpoint** node.



For more information about working with the Test Results, see Chapter 16, “Viewing Test Results.”

28

Creating a Service Emulation

The Service Emulation tool lets you create an emulation of a Web service for the purpose of testing.

The following information only applies to Web Services and SOA Vuser scripts.

About Creating a Service Emulation

HP Service Test provides a Service Emulation tool for creating an emulation of a service in order to test other Web services in your environment.

The emulated service lets you design and run tests at early stages of development when the actual service is inaccessible. For example if the development of the service is incomplete or if the service's host is unavailable, you can use an emulated service to test other services in your application.

In the Service Emulation interface, you associate a WSDL document to the service which indicates its operations and parameters. You specify rules and delays to define the service's behavior.

The steps in creating an emulated service are:

- Starting the Emulation Server
- Selecting a Host
- Adding a New Service
- Setting the Emulated Service's Behavior
- Manipulating Emulated Services

Starting the Emulation Server

The Service Emulation tool provides a Tomcat server through which you can run emulated services. The installation program installs an Axis servlet which allows you to run the service on the Tomcat server running on the local machine.

To create or run emulated services on your local machine, you must manually start the emulation service.

To start the emulation service:

- Choose **Start** > **Programs** > **HP Service Test** > **Start Emulation Service**.

To stop the emulation service:

- Choose **Start** > **Programs** > **HP Service Test** > **Stop Emulation Service.**

You can also manually start and stop the service from the Services dialog box (**Start Programs** > **Administrative Tools** > **Services**). The service is called **Mercury ServiceEmulation**.

To check whether or not the server is active, enter the following URL into your browser: http://localhost:8080/ServiceEmulation/index.jsp.

Troubleshooting the Server

If the Service Emulation console indicates that the server is not accessible, even after a manual start, you can try the following troubleshooting tips:

- Make sure the server is up. Enter the following URL into your browser: http://localhost:8080/ServiceEmulation/index.jsp. If the server is down, start it manually from the Start menu.
- Verify that port 8080 is available. If it is not, release the port and restart the server.
- Open the Apache Tomcat log files under the <product install dir>\Service Test\apache-tomcat-5.5.17\logs directory and determine the reason that the server did not load. Fix the problem and reload the server.
- When emulating a service, if you encounter a warning "Service is not activated" in the endpoint location of your emulated service, perform one or more of the following actions:
  - Verify that the service is active. Select the service in the left pane and click the Activate Service button in the right pane, if it is visible.
  - If the service is already active, check its URL. Copy the URL from the WSDL Location in the Emulated Service's right pane, and paste it into a browser, removing the suffix **?wsdl** from the string. For example, instead of http://localhost:8080/axis/services/MyService?wsdl, use http://localhost:8080/axis/services/MyService. If your browser opens a valid page, then your service is active. To use this emulated service, import the original WSDL into Service Test. Then override the address and set it to the URL you previously used in the browser.

    For more information about integrating your emulated service into your test, see "Using Emulated Services in Vuser Scripts" on page 469.

Selecting a Host

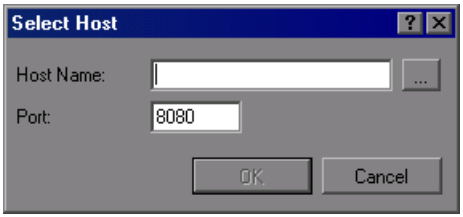
The first step in creating an emulated service is choosing a host for the emulated services. The setup installs a Tomcat server on your local machine. You can also specify external servers, upon which the Tomcat server is installed.

By default, the Service Emulation tool uses localhost as the emulation server.

To add an external host:

1. Open the Service Emulation tool. Choose **Start** > **Programs** > **HP Service Test** > **Service Emulation Console**.

2. Choose **Main** > **New Host** or click the **Add Host** button to open the Select Host dialog box.



3. Browse for a host in your network, or enter the host name directly. If applicable, change the port setting from the default value, 8080.
4. Click **OK**. The host is added to the list in the window ‘s left pane.

Adding a New Service

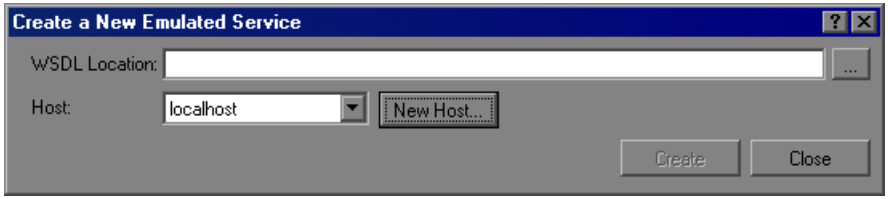
For each host that you added, you can create emulated services.

You create a service by specifying a WSDL file that defines the operations and parameters of the service. When you specify a WSDL file, the Service Emulation tool uses the current structure of the WSDL to define the structure of the service’s input and output data.

If the original WSDL changes, it will not be reflected in your emulated service. To use an updated WSDL, recreate the emulated service.

To define a new service:

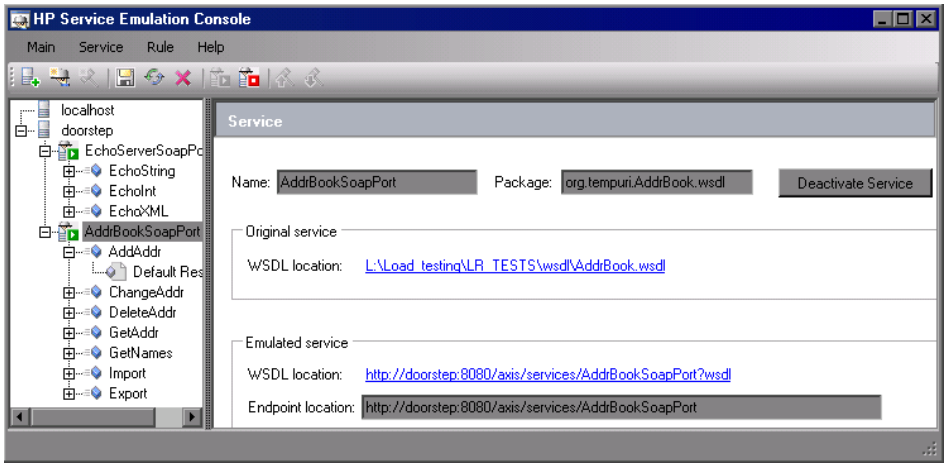
1 Choose **Main** > **New Emulated Service** or click the **New Emulated Service** button to open the Emulate New Service dialog box.



2 Enter the full path or URL of the WSDL in the **WSDL Location** box, or click the Browse button to the right of the box, to locate the WSDL.

3 Choose a host from the list, or click **New Host** to add a new one.

4 Click **Create** to add the new service. All services are listed in the window's left pane.



5 To delete a specific service, select it in the left pane and choose **Delete** from the right-click menu.

Setting the Emulated Service's Behavior

The Service Emulation lets you specify a behavior for each of the operations defined in the WSDL. You specify the behavior of the service's operations by:

- Providing the Default Response
- Setting Rules

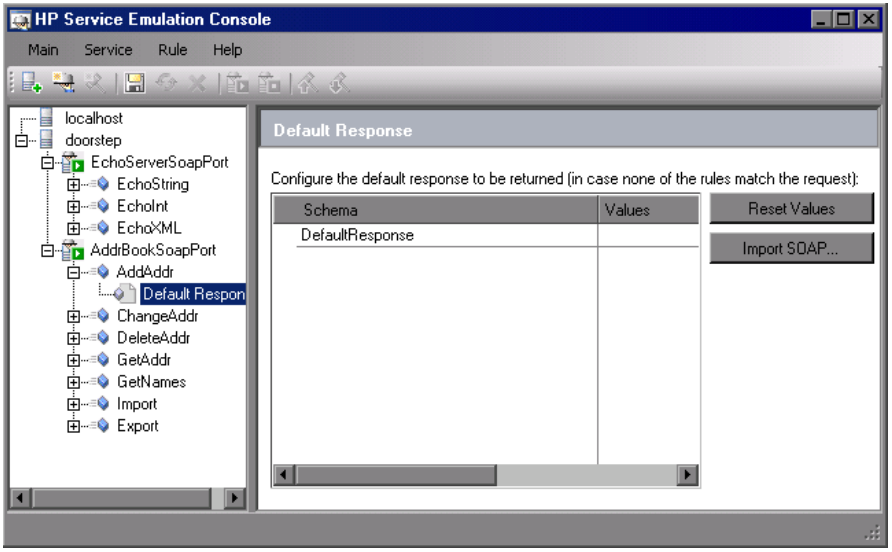
Providing the Default Response

The default response is the default values that you expect the operation to provide.

You can manually specify the values for the default response, or you can import them from an XML file containing sample results.

To specify a default response:

1. Expand the service and desired operation in the left pane.
2. Select the **Default Response** node of your service. The right pane shows a Default Response table.



3. In the **Values** column, specify a default response value.

4. To import values from the SOAP response, click **Import SOAP**.
5. Select the XML file with a valid SOAP response and click **OK**.
6. To instruct the service to use the default response values, click **Reset Values**.

Setting Rules

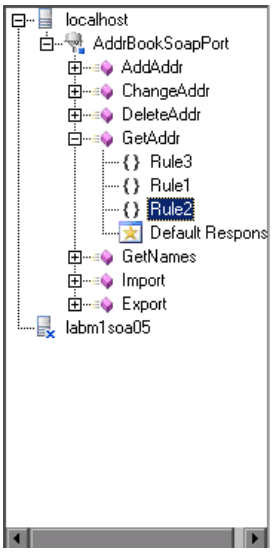
In addition to setting the default response, you can also set behavior rules. Through rules, you define a distinct behavior for the service—the expected response based on the request, input data.

You can set the rules manually, or by importing them from an existing XML SOAP file.

To set rules manually, you insert values for one or more of the arguments and the corresponding response. For example, for an Add operation, you could specify 4 as the first argument, 5 as the second argument, and a result of 9.

If necessary, you can exclude a specific argument This is useful if you want your service to return a particular response, ignoring the value of one of the arguments. For example, for a multiplication operation, you can set a rule indicating that if the first argument is 0, the result will be 0, regardless of the value of the second argument.

You can set multiple rules for your operation. You place the rules in order of priority—the higher rules have priority over the lower ones in case of a conflict. In the following example, Rule 3 has the highest priority.



To set a rule manually:

1. Expand the service and desired operation in the left pane.
2. Select the operation in the left pane and choose **Add New Rule** from the right-click menu or click the **New Rule** button.
3. Specify values for the request and response.
4. Select the check boxes adjacent to each of the arguments that you want to impact the rule.
5. Click **Save** to save all of the modifications you made to the service.
6. To import values from SOAP file, click **Import SOAP** and specify an XML file containing the SOAP message. You can import SOAP messages for both the request and the response.
7. For additional rules, repeat the above steps.
8. To move a rule up to raise its priority, use the toolbar button or right-click and choose **Move up**.

To move a rule down, use the toolbar button or right-click and choose **Move down**.

Manipulating Emulated Services

After you create your emulated services you can manipulate them by:

- Reloading an Emulated Service
- Inserting Delays for Operations
- Deactivating and Activating a Service
- Changing Port Numbers

Reloading an Emulated Service

Reloading emulated services reverts to the last saved version of the service on the host. Any rules which were not saved will be lost. If another user saved changes to the service from a different machine, by reloading the service, you will bring those changes on to your machine.

This feature is useful if you made changes and you want to discard them and revert back to the saved version. Another use for this feature is if another user modified the service behavior, you can reload the service to get their changes.

To reload a service:

1. Select the service you want to reload.

2. Click the **Reload** button or choose **Reload** from the right-click menu.

Inserting Delays for Operations

To more accurately emulate a service, you can insert a delay time for the response of each operation. This delay emulates a delay in the response of the server after the application submitted a request.

To insert a delay:

1. Select an operation in the left pane.

2 In the right pane, enter the delay in seconds.

Deactivating and Activating a Service

After you create a service, you can deactivate it temporarily instead of deleting it. If you deactivate a service, it will be ignored in your tests, but it will still be available if you need to implement it in the future.

In the list of emulated services, a green box indicates that a service is active and a red box indicates that it not active.

To deactivate a service, select it in the left pane and click the **Deactivate** button or click **Deactivate Service** in the right pane.

To activate a service that was previously deactivated, select it in the left pane and click the **Activate** button or click **Activate Service** in the right pane.

Changing Port Numbers

You can modify the port of your emulated service by changing the port number in the emulation's configuration file. The configuration file, httpd.conf, is stored in Service Test's **apache/conf** directory.

To modify the port number:

1 Open the **httpd.conf** file with any text editor.

2 Modify the entry "Listen 80" to the desired port number, for example "Listen 8080".

3 Modify the entry which contains "ServerName localhost:80" to indicate the desired port, for example "ServerName localhost:8080".

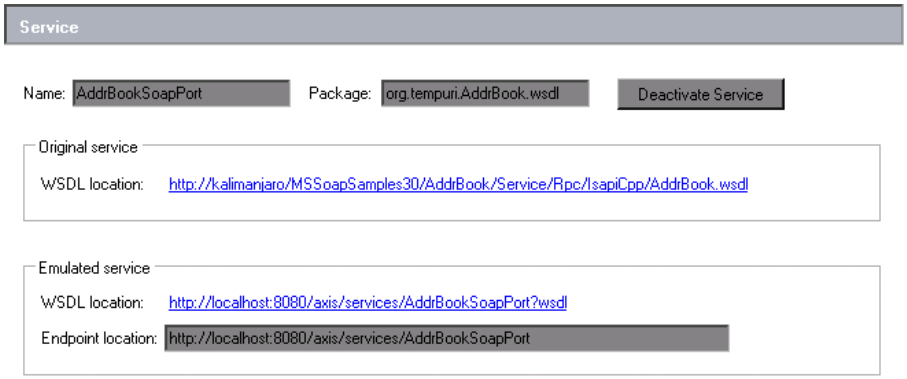
4 Restart the Apache server. Select **Programs** > **Start** > **HP Service Test** > **Start Emulation Server**.

Using Emulated Services in Vuser Scripts

After you create an emulated service, you can import it into your Vuser script for testing purposes.

To use an emulated service:

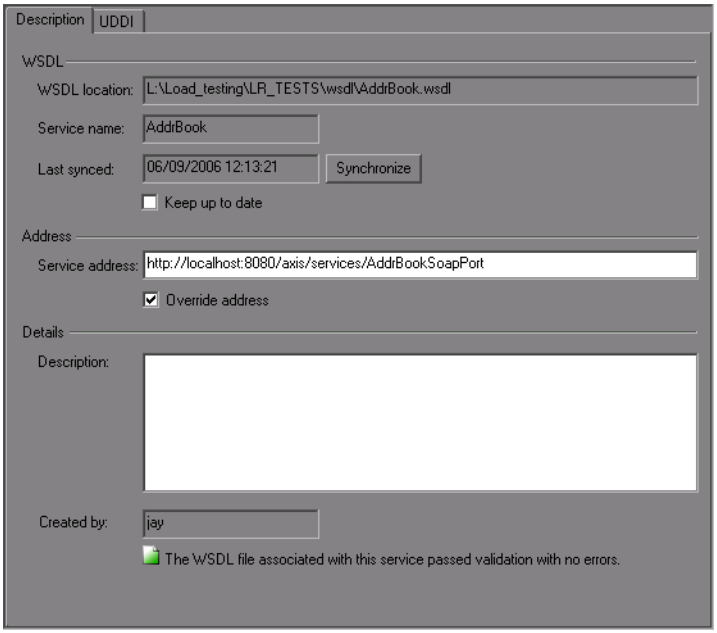
1 In the Service Emulation console, select a service in the left pane and copy the **Endpoint location** from the right pane to the clipboard.



2 In Service Test, Open the Service Management window. Select **SOA Tools** > **Manage Services** or click the Service Management toolbar button.

3 If you have not already imported the service you want to emulate, click **Import** to import a WSDL. If you have already imported the service that you want to emulate, select it in the left pane.

4 Enable the **Override address** option.



5 Paste the Endpoint location into the **Service Address** box. During the test run, the service will respond to that location.

Part IV

Information for Advanced Users

29

Creating Vuser Scripts in Visual Studio

You can create a Vuser script template in Visual Studio using Visual C or Visual Basic. You compile it as you would a regular C or Visual Basic program.

About Creating Vuser Scripts in Visual Studio

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 UNIX 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 UNIX—recording is not supported on UNIX.

Visual Studio

For users working with Visual Studio, you can program in Visual Basic, 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 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 "Creating Custom Vuser Scripts" in Volume II - the *Protocols user guide*.

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

After creating a basic Vuser script from a template, you can enhance the script to provide run-time information and statistics. For more information, see Chapter 8, "Enhancing Vuser Scripts."

An online C reference of the common functions used in Vuser scripts, are included in the *Online Function Reference* (**Help** > **Function Reference**).

Creating a Vuser Script with Visual C

Please note that you can create Vuser scripts using Visual C version 6.0 or higher.

To create a Vuser script with Visual C:

1. In Visual C, create a new project - dynamic link library (dll). Choose **File** > **New** and click the Projects tab.
2. In the Wizard, choose *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 choose **Code generation** (Category) > **Use Run Time library** (List). Change it to: **Multithreaded dll.**
- Select the C/C++ tab and choose **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, refer to the General functions in the *Online Function Reference* (**Help** > **Function Reference**).

7 Build the project. The output will be a DLL.

8 Create a directory with the same name as the DLL and copy the DLL to this directory.

9 In the **lrvuser.usr** file in the *Template* directory, Update the USR file key *BinVuser* with the DLL name: BinVuser=<*DLL_name*>.

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
```

Creating a Vuser Script with Visual Basic

To create a Vuser in Visual Basic:

1. In Microsoft Visual Basic, create a new project. Select **File** > **New Project**.
2. Select **LoadRunner Virtual User**. A new project is created with one class and a template for a Vuser.
3. Save the project before you continue to program. Chose **File** > **Save Project**.

4 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, Vuser_Init, Vuser_Run, and Vuser_End.

```
Option Explicit

Implements Vuser

Private Sub Vuser_Init()
'Implement the Vuser initialization code here
End Sub

Private Sub Vuser_Run()
'Implement the Vuser main Action code here
End Sub

Private Sub Vuser_End()
'Implement the Vuser termination code here
End Sub
```

5 Add code from your client application, or program as you normally would.

6 Use the Object Browser to add the desired VuGen elements to your code, such as transactions, think time, rendezvous, and messages, using the object browser.

7 Enhance your program with run-time settings and parameters. For more information, see "Configuring Runtime Settings and Parameters" on page 478.

8 Build the Vuser script: select **File** > **Make** *project_name*.dll.

The project is saved in the form of a Vuser script (.usr). The script resides in the same directory as the project.

Configuring Runtime 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* directory of the product installation.

To configure runtime settings and parameterize scripts:

1. In the product's *bin* directory, double-click on *lrbin.bat*. The Standalone Vuser Configuration dialog box opens.

2. Choose **File** > **New**. Specify a script name for the *usr* file. The script name must be identical to the name of the directory to which you saved the DLL.
3. Choose **Vuser** > **Advanced** and enter the DLL name in the Advanced dialog box.
4. Choose **Vuser** > **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 Chapter 13, "Configuring Run-Time Settings."
5. Choose **Vuser** > **Parameter List** to define parameters for your script. The Parameter dialog boxes are identical to those in VuGen. For more information, see Chapter 9, "Working with VuGen Parameters."

   Test the script by running it in standalone mode. Choose **Vuser** > **Run Vuser**. The Vuser execution window appears while the script runs.
6. Choose **File** > **Exit** to close the configuration utility.

30

Programming with the XML API

You can create Vuser scripts that support the complete XML structure. VuGen provides functions that allow you to query and manipulate the XML data.

The following information applies primarily to Web, Web Services, and Wireless Vuser scripts.

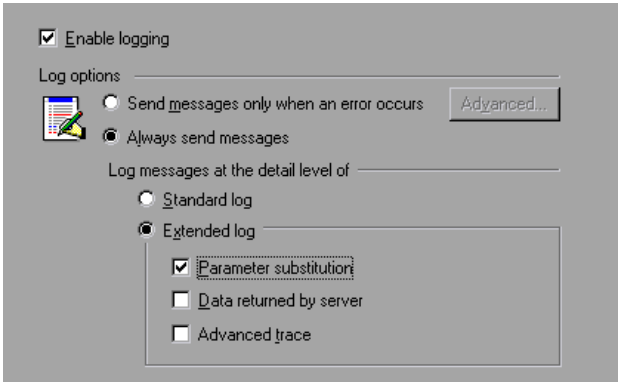
About Programming with the XML API

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 choose **Parameter Substitution**. For more information, see Chapter 13, "Configuring Run-Time Settings."



All Vuser API XML functions return the number of matches successfully found, or zero for failure.

Understanding XML Documents

XML, or Extensible Markup Language, is a markup language that you can use to create your own custom tags. Using these tags, you give a meaning to the text between the tags. This stands in contrast to standard HTML tags such as H1, P, DIV, and so on, which cannot be customized and do not indicate the content of the text.

XML documents consist of trees with many nodes and branches. There are three common terms used that describe the parts of an XML document: tags, elements, and attributes. The following example illustrates these terms:

```
<acme_org>
    <accounts_dept>
        <employee type='PT'>
            <name>John Smith</name>
            <cubicle>227</cubicle>
            <extension>2145</extension>
        </employee>
    </accounts_dept>
    <engineering_dept>
        <employee type='PT'>
            <name>Sue Jones</name>
            <extension>2375</extension>
        </employee>
    </engineering_dept>
</acme_org>
```

A *tag* is the text between the left and right angle brackets. <acme_org>, <employee> and <name> are examples of tags. There are starting tags, such as <name>, and ending tags, such as </name>. The above XML fragment describes the Acme organization with two employees, John Smith and Sue Jones.

An *element* is the starting tag, ending tag, and everything in between. In the sample above, the <employee> element contains three child elements: <name>, <cubicle>, and <extension>.

An *attribute* is a name-value pair inside the starting tag of an element. In this example, **type='PT'** is an attribute of the <employee> element;

In the above example, the tag *name* is an element of *employee*. Each element has a value. An example of a *name* element's value is the string "John Smith".

Using XML Functions

The next sections provide 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. These 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);
```

Working with 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 an XML Script

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

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;
}
```

Using 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.

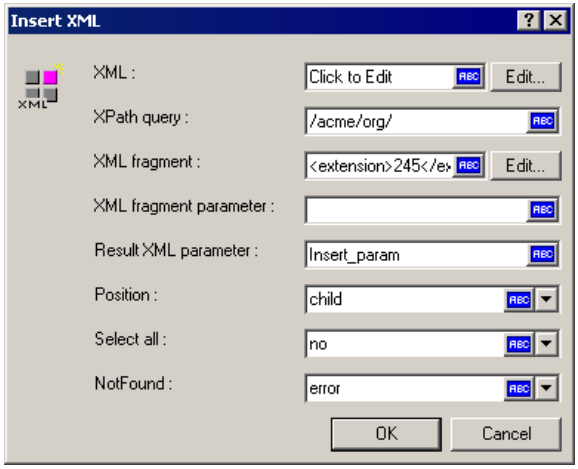
The functions that support result parameters are **lr_xml_insert**, **lr_xml_transform**, **lr_xml_replace**, **lr_xml_delete**, and **lr_xml_set_values**.

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.

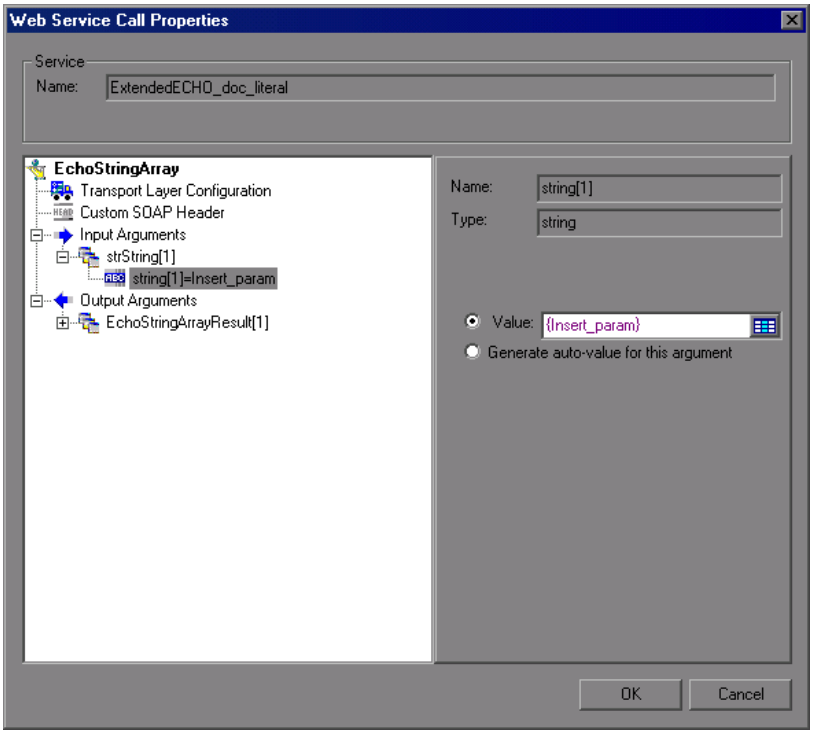
To use the result parameter as input:

1. In Tree view, 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 "Working in the Web Service Call View" in *Volume I-Using Service Test*.

31

VuGen Debugging Tips

You can use the following integration and configuration tips to help you produce an error-free Vuser script.

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 */
```

Adding 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.

To 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.

Debugging Database Applications

The following tips apply to database applications only (Oracle, ODBC, Ctlib):

- Generating Debugging Information
- Examining Compiler Information
- Code Generation Information
- Preprocessing and Compilation Information

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 directory 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 directory. 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.

Working with 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 Vusers 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.

Solving Common Problems with 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 directory. 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 UNIX, 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
```

Two-tier Database Scripting Tips

The following section offers solutions for two-tier database scripts. For Siebel specific solutions, see "Siebel-specific Scripting Tips" on page 512.

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, it is recommended 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 like

```
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, 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 choose 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);
```

Siebel-specific Scripting Tips

This section offers solutions for Siebel database users. You should also see the previous section which discusses some general database scripting tips.

Question 9: Virtual users run fine in VuGen but fail in the Controller with duplicate key violations.

Answer: The Siebel client stores a key in the NEXT_SUFFIX column of the S_SSA_ID table. This client has code that detects and recovers from situations in which it fails to successfully get a block of suffix values.

VuGen automatically correlates the NEXT_SUFFIX and MODIFICATION_NUM fields of the S_SSA_ID table. During an UPDATE the MODIFICATION_NUM field is incremented by 1 and the NEXT_SUFFIX field is increased by 100 in base 36. However, VuGen does not add code in instances where a client could not obtain a new block of suffix values. As a result, the replay fails with a unique constraint error, when you attempt to insert new values into the database.

You must manually add code to each location in the script where a block of suffixes is obtained, in order to perform a retry if the first attempt fails. You can locate these places by searching for SiebelPreSave in the script. You must also add a *while* loop with code similar to the example below. This example only works for Oracle. For ODBC use **lrd_row_count** instead of using the fourth argument of **lrd_exec**.

```
unsigned long lRowUpdated;
int nAttempt;

…

// This loops until we successfully obtain a “next_suffix”
lRowUpdated = 0;
nAttempt=0;

while (lRowUpdated != 1) {

   nAttempt++;
   if (nAttempt > 1)
      lr_output_message (".......Next suffix retry %d", nAttempt);
   else
   {
      lrd_open_cursor(&Csr13, Con1, 0);
      lrd_stmt(Csr13, "SELECT\n T1.LAST_UPD,\n T1.CREATED_BY,\n "
         "T1.CONFLICT_ID,\n T1.CREATED,\n T1.NEXT_SUFFIX,\n "
         "T1.ROW_ID,\n T1.NEXT_PREFIX,\n T1.CORPORATE_PREFIX,\n "
         "T1.MODIFICATION_NUM,\n T1.NEXT_FILE_SUFFIX,\n      "
         "T1.LAST_UPD_BY\n   FROM \n SIEBEL.S_SSA_ID T1", -1, 1, 1, 0);
   }
   lrd_bind_cols(Csr13, BCInfo_D375, 0);
   lrd_exec(Csr13, 0, 0, 0, 0, 0);
```

```
    SiebelPreSave_1();
    lrd_fetch(Csr13, -1, 4, 0, PrintRow26, 0);
    GRID(26);
    SiebelPostSave_1();

    if (nAttempt > 1)
    {
     lrd_open_cursor(&Csr14, Con1, 0);
     lrd_stmt(Csr14,"\nUPDATE SIEBEL.S_SSA_ID SET\n LAST_UPD_BY=:1,\n "
     "NEXT_SUFFIX = :2,\n      MODIFICATION_NUM = :3,\n      LAST_UPD = "
     ":4\n WHERE\n ROW_ID = :5 AND MODIFICATION_NUM = :6\n", -1, 1,
     1, 0);
    }
    lrd_assign_bind(Csr14, "6", "<modification_num>", &_6_D376, 0,
                LRD_BIND_BY_NUMBER, 0);
    lrd_assign_bind(Csr14,"5", "0-11",&_5_D377,0,LRD_BIND_BY_NUMBER, 0);
    strcpy (szTimeAtNewButton, lr_eval_string("<Now>"));
    sprintf (szTimeStamp, "%s %s", lr_eval_string("<Today>"),
            szTimeAtNewButton);
    lr_save_string (szTimeStamp, "DateTimeStamp");
lrd_assign_bind(Csr14, "4", "<DateTimeStamp>", &_4_D378, 0,
       LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "3", "<next_modnum>", &_3_D379, 0,
       LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "2", "<next_suffix_x100>", &_2_D380, 0,
       LRD_BIND_BY_NUMBER, 0);
lrd_assign_bind(Csr14, "1", "1-1E1",&_1_D381,0,LRD_BIND_BY_NUMBER, 0);

// this update won't update any rows unless we successfully got our suffix
lrd_exec(Csr14, 1, 0, &lRowUpdated, 0, 0);
lrd_commit(0, Con1, 0);

}//while
      lr_output_message ("…Rows updated %ld", lRowUpdated);
```

Question 10: How can I find the correct value to correlate for a primary key?

Answer: Siebel tends to generate key values based on base 36 mathematical manipulations of *<next_suffix>*. Try comparing several recordings and try to determine the relationships. You can ignore date fields when correlating Siebel, since they do not seem to effect script replay.

Question 11: How can I solve an INSERT into S_SRV_REQ failure with a duplicate key violation?

Answer: The primary key is SR_NUM. Newer versions of VuGen automatically correlate insertions into this table, by using the function lrd_siebel_str2num, which converts the NEXT_SUFFIX value of the S_SSA_ID table from base 36 to the base 10 equivalent. Older versions of VuGen might not handle this correlation correctly.

Question 12: VuGen does not automatically perform all the correlations I need in order to replay my script correctly. How can I add the missing correlations?

Answer: Currently VuGen only saves the values of the NEXT_SUFFIX and MODIFICATION_NUM columns from the S_SSA_ID table and replaces them with parameters when they are used later in the script. You may need to add some additional correlations manually. The correlation code in the **SiebelPreSave** and **SiebelPostSave** functions in the *print.inl* file can serve as an example of how to correlate specific values once you determine what needs to be correlated.

- Sometimes the NEXT_FILE_SUFFIX and MODIFICATION_NUM columns are chosen from the S_SSA_ID table. In this case, an UPDATE statement updates the NEXT_FILE_SUFFIX by adding one to this string in base 36, and one to the MODIFICATION_NUM. The value of the NEXT_FILE_SUFFIX will often be inserted in the FILE_REV_NUM field of a table. Often the name of this table ends with the *_ATT* suffix, to indicate that it is an attachment.
- Whenever Siebel performs an UPDATE statement, there is a MODIFICATION_NUM column that is incremented by one. VuGen only generates this correlation automatically for the S_SSA_ID table. You have to do it manually for other cases.

- Siebel refers to records according to their ID number. Siebel usually finds all records of a particular type (such as an agreement), and then later uses the ID number for a record when trying to update or delete an existing record of this type. You need to replace the ID number by a parameter during replay in order to generate a meaningful load test. The ID number has the form of one or more digits, a hyphen, followed by one or more alphanumeric characters, such as 1-QPF9. VuGen does not do this parameterization automatically, so you have to do it manually.
- If you find any other missing correlations or parameterizations, please notify customer support in order that HP can improve VuGen's support for Siebel.

Running PeopleSoft-Tuxedo Scripts

To run PeopleSoft-Tuxedo Vusers with Tuxedo 7.x, you must change the library extension in the *mdrv.dat* file:

```
[PeopleSoft-Tuxedo]
WINNT_EXT_LIBS=lrt7.dll
```

32

Advanced Topics

The following advanced information can assist you in determining the replay issues and debugging your Vuser script.

Files Generated During Recording

Assume that the recorded test has been given the name 'vuser' and is stored under c:\tmp. Following is a list of the more important files that are generated after recording:

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 directory stores all of the recorded data used primarily as a backup. Once the data is in this directory, 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
AppName=C:\PROGRA~1\Netscape\COMMUN~1\Program\netscape.exe
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
```

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.c
   ```

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\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"
```

Running a Vuser from the Unix Command Line

VuGen includes a Unix 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 Unix.

Place the file *run_db_Vuser.sh* in the $M_LROOT/bin directory. 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 prepocessing 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" on page 524.

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 Unix 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 Unix utility, *run_db_vuser*, does not yet support this option.

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 knowledgebase 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.

To determine its full command line:

1. Rename *MyOleSrv.exe* to *MyOleSrv.orig.exe*.
2. Place *CmdLine.exe* in the same directory 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” on page 526 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).

Examining the .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 directory 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* directory. Each protocol has a subdirectory 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 directory 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 directory.

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. The next section describes what you need to add to this file in order to define a new protocol.

Adding a New Vuser Type

To add a new Vuser type/protocol to VuGen, you need to:

- Edit the *mdrv.dat file* with the new protocol's settings.
- Add a *.cfg* file.
- Insert an *.lrp* file.
- Create a template directory.

Editing the mdrv.dat File

First, you edit the mdrv.dat file which resides in the M_LROOT\dat directory. You 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
```

Adding a CFG file

You can optionally 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 subdirectory 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
```

Inserting an LRP file

In the dat/protocols directory, 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 directory as a base for your new protocol.

Specifying a Template

After adding an *lrp* file, insert a subdirectory to *M_LROOT/*template with a name corresponding to the protocol name defined in the *lrp* file. In this subdirectory, 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 subdirectory contains a file called *globals.h* which directs VuGen to the *as_web.h* file in the include directory:

```
#include #as_web.h"
```

Part V

Appendixes

A

Calling External Functions

When working with VuGen, 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:

- locally—for one script, using the **lr_load_dll** function
- globally—for all scripts, by adding statements to the vugen.dat file

Loading a DLL—Locally

You use the **lr_load_dll** function to load the DLL in 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.

To 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 UNIX 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 UNIX 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.

Loading a DLL—Globally

You can 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.

To call a function defined in a DLL:

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* directory.

   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 Winsocket 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.

B

Working with Foreign Languages

VuGen supports multilingual environments, allowing you to use languages other than English on native language machines when creating and running scripts.

About Working with Foreign Languages

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.

Manually Converting String Encoding

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. The syntax of the function is:

```
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 Execution 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\x00"
Ending action Action.
```

The result of the conversion is saved to the *paramName* argument.

Converting String Encoding In Parameter Files

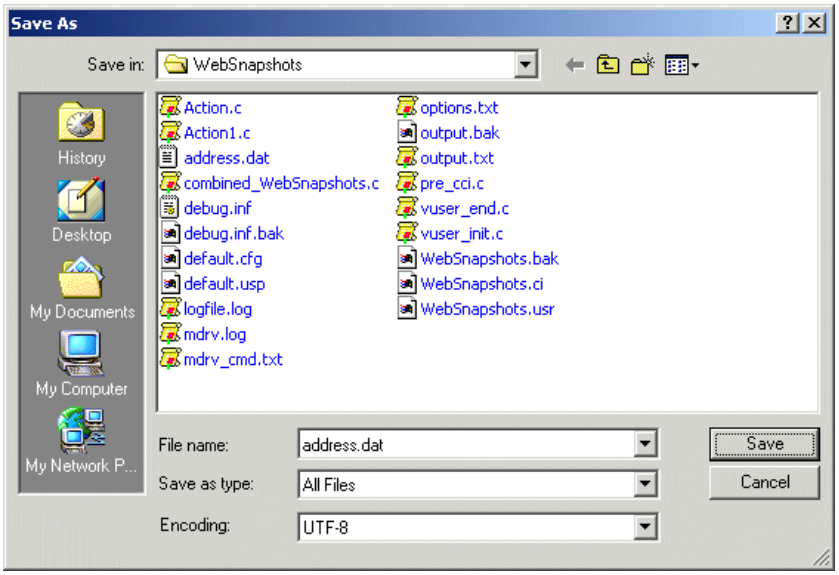
The parameter file contains the data for parameters that were defined in the script. This file, stored in the script’s directory, has a **.dat* extension. When running a script, Vusers use the data to execute actions with varying values.

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.

To apply UTF-8 encoding to a parameter file:

1. Choose **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.

Setting the String Encoding for Web Record and Replay

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 section discusses:

- Encoding Recording Option
- Manually Enabling Encoding
- Browser Configuration

Encoding Recording Option

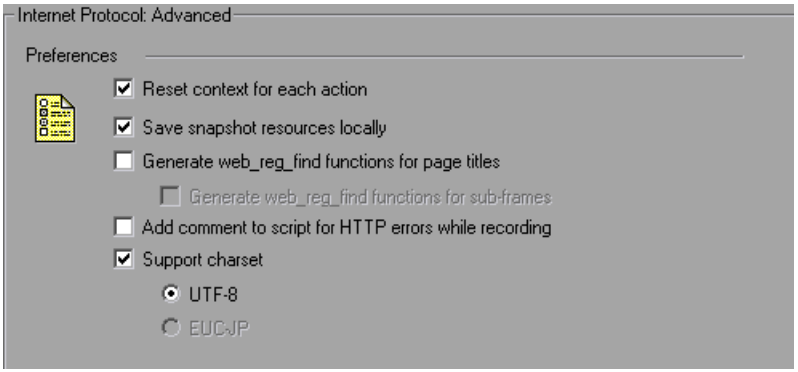
In order to be recognized as a non-English Web page, the page must indicate the charset in the HTTP header or in the HTML meta tag. Otherwise, VuGen will not detect the EUC-JP encoding and the Web site will not be recorded properly. To instruct VuGen to record non-English requests as **EUC-JP** or **UT-8**, select the appropriate option in the Recording Options dialog box, **HTPP Properties: Advanced** node.

- **UTF-8.** This option 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.
- **EUC-JP.** For users of 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 machine, and adds a **web_sjis_to_euc_param** function to the script (Kanji only).

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.

To enable the charset Encoding:

1 Open the Recording Options (Ctrl+F7) and select the **Advanced** node.



2 Select **Support charset**. Choose the desired character encoding—UTF-8 or EUC-JP (only enabled for the Kanji operating system).

3 Click **OK**.

For more information about these settings, see "Setting Recording Options for Internet Protocols" in the *VuGen Protocols* guide.

Manually Enabling Encoding

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**.

Browser Configuration

If, during recording, non-English characters in the script are displayed as escaped hexadecimal numbers (For example, the string "Ü&" becomes "%DC%26"), you can correct this by configuring your browser not to send URLs in UTF-8 encoding. In Internet Explorer, choose **Tools** > **Internet Options and** click the **Advanced** tab. Clear the **Always Send URLs as a UTF-8** option in the Browsing section.

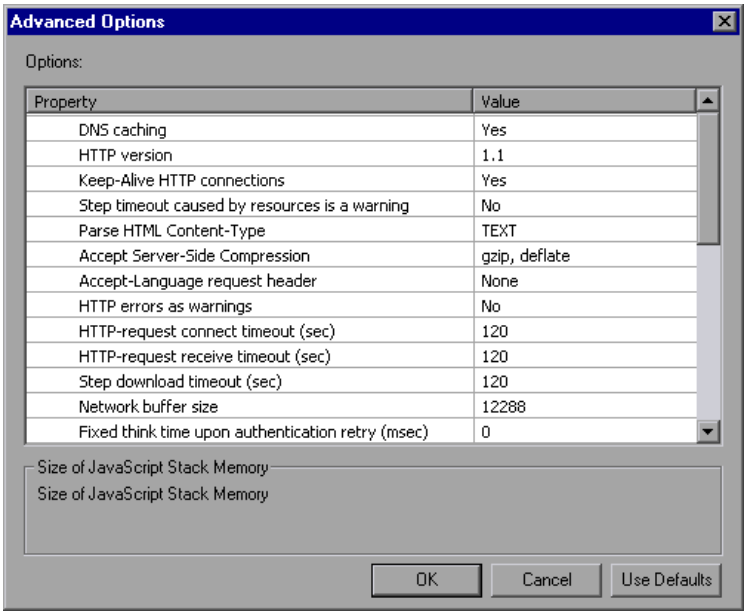
For more information about **web_sjis_to_euc_param**, refer to the *Online Function Reference*.

Specifying a Language for the Accept-Language Header

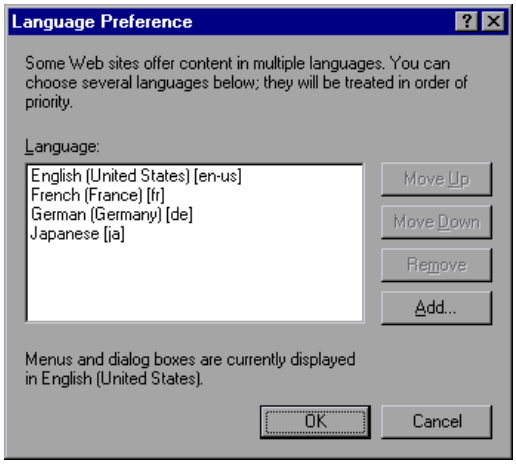
Before running a Web script, you can set the page's request header to match your current language. In the Internet Protocol Run-Time settings, you set the language of the *Accept-Language* request header. This header provides the server with a list of all of the accepted languages.

To set the Accept-Language header:

1. Open the Run-Time settings (F4) and select the **Internet Protocols:Preferences** node.
2. In the Advanced section, click the **Options** button to open the Advanced Options dialog box.



3 Locate the **Accept-Language request header** option. In the **Value** column, select the desired language from the list. This list is derived from the Internet Options Language settings in your browser.



For more information about these settings, see "Configuring Internet Run-Time Settings" in the *VuGen Protocols* guide.

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, REAL or VB in VBA mode, the length of the script name is limited to 10 multi-byte characters (21 bytes).

Quality Center Integration

To open a script saved in a Quality Center project from VuGen, or a scenario saved in a Quality Center project from Controller, add a new Test Set named "Default" (in English) to the Quality Center project.

C

Programming Scripts on UNIX Platforms

Vusers on UNIX platforms can create scripts through programming. To create a script through programming, you use a template.

About Programming Vuser Scripts to Run on UNIX Platforms

There are two ways to create Vuser scripts that run on UNIX platforms: by using VuGen, or by programming.

VuGen	You can use VuGen to create Vuser scripts that run on UNIX platforms. You record your application in a Windows environment and run it in UNIX—recording is not supported on UNIX.
programming	Users working in UNIX-only environments can program Vuser scripts. Scripts can be programmed in C or C++ and they must be compiled into a dynamic library.

This appendix describes how to develop a Vuser script by programming.

To create a script through programming, you can use a Vuser template as a basis for a larger Vuser scrips. The template provides:

- correct program structure
- Vuser API calls
- source code and makefiles for creating a dynamic library

After creating a basic script from a template, you can enhance the script to provide run-time Vuser information and statistics. For more information, see Chapter 8, “Enhancing Vuser Scripts.”

Generating Templates

VuGen includes a utility that copies a template into your working directory. The utility is called mkdbtest, and is located in $M_LROOT/bin. You run the utility by typing:

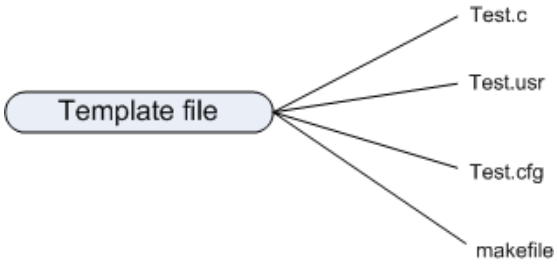
```
mkdbtest name
```

When you run mkdbtest, it creates a directory called name, which contains the template file, name.c. For example, if you type:

```
mkdbtest test1
```

mkdbtest creates a directory called *test1*, which contains the template script, *test1.c*.

When you run the *mkdbtest* utility, a directory is created containing four files *test.c*, *test.usr*, *test.cfg* and *Makefile*, where *test* is the test name you specified for mkdbtest.



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 UNIX signals. Do not use these signals in your Vuser programs.

Configuring Vuser Run-Time Settings

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 Chapter 13, "Configuring Run-Time Settings".) 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 options for Unix 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"
```

Defining Transactions and Rendezvous Points

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.

Compiling Scripts

After you modify the template, you compile it with the appropriate *Makefile* in the script's directory. 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 directory 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 UNIX command line, type:

```
mdrv -usr 'pwd' test.usr
```

where *pwd* is the full path to the directory 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 profile. For more information, refer to the *HP LoadRunner Controller*, *Performance Center*, or *HP Business Availability Center* documentation.

D

Using Keyboard Shortcuts

The following list describes the keyboard shortcuts available in the Virtual User Generator.

ALT+F8	Compares the Current Snapshots (Web Vusers only)
ALT+INS	Create New Step
CTRL+A	Select All
CTRL+C	Copy
CTRL+F	Find
CTRL+G	Go To Line
CTRL+H	Replace
CTRL+N	New
CTRL+O	Open
CTRL+P	Print
CTRL+S	Save
CTRL+V	Paste
CTRL+X	Cut
CTRL+Y	Redo
CTRL+Z	Undo
CTRL+F7	Recording Options
CTRL+F8	Scan for Correlations

CTRL+SHIFT+SPACE	Show Function Syntax (Intellisense)
CTRL+SPACE	Complete Wizard (completes the function name)
F1	Help
F3	FIND Next Downward
SHIFT+F3	Find Next Upward
F4	Run-Time Settings
F5	Run Vuser
F6	Move Between Panes
F7	Show EBCDIC Translation Dialog (for WinSocket data)
F9	Toggle Breakpoint
F10	Run Vuser Step by Step

E

Recording Options and Run-Time Settings

The following tables provide an alphabetical and categorical index for the recording options and run-time settings in Service Test.

Run-Time Settings

The run-time settings let you set the behavior of Vusers during replay of the script. If you are running the Vusers through the LoadRunner Controller, the run-time settings control the Vusers' behavior in the scenario.

To learn more about a run-time setting, locate it in one of the tables and refer to the appropriate section as shown in the right column. You can use either of the listings:

- Alphabetical Listing of Run-Time Settings
- Categorical Listing of Run-Time Settings

Alphabetical Listing of Run-Time Settings

Run-Time Setting	Category	For more information, see:
Additional Attributes	General	"Configuring Additional Attributes Run-Time Settings" on page 233
Advanced	JMS	"Setting Web Services JMS Run-Time Settings" *in Volume II - the Protocols user guide*

Browser Emulation	Browser	"Setting Browser Emulation Properties" in *Volume II - the Protocols user guide*
Classpath	Java Environment Settings	"Setting the Run-Time Classpath Options" in *Volume II - the Protocols user guide*
Client Emulation	Oracle NCA	"Configuring Oracle NCA Run-Time Settings" in *Volume II - the Protocols user guide*
Configuration	Citrix	"Citrix Configuration Run-Time Settings" in *Volume II - the Protocols user guide*
Content Check	Internet Protocol	"About Checking Web Page Content" in *Volume II - the Protocols user guide*
Download Filters	Internet Protocol	"Filtering Web Sites" in *Volume II - the Protocols user guide*
Gateway	WAP	"Configuring Gateway Options" in *Volume II - the Protocols user guide*
General	SAPGUI	"Setting SAPGUI Run-Time Settings" in *Volume II - the Protocols user guide*
Java VM	Java Environment Settings	"Specifying the JVM Run-Time Settings" in *Volume II - the Protocols user guide*
Log	General	"Configuring the Log Run-Time Settings" on page 226
Miscellaneous	General	"Configuring Miscellaneous Run-Time Settings" on page 234
.NET Environment	.NET	"Configuring .NET Environment Run-Time Settings" *in Volume II - the Protocols user guide*
Pacing	General	"Configuring Pacing Run-Time Settings (multi-action)" on page 223
Preferences	Internet Protocol	"Setting Internet Preferences" in *Volume II - the Protocols user guide*

Proxy	Internet Protocol	"Setting Proxy Options" in *Volume II - the Protocols user guide*
Radius	WAP	"Configuring Radius Connection Data" in *Volume II - the Protocols user guide*
RTE	RTE	"Configuring RTE Run-Time Settings" in *Volume II - the Protocols user guide*
Run Logic	General	"Setting Pacing and Run Logic Options (single action)" on page 224
Server and Protocol	MMS (Multimedia Messaging Service)	"Configuring MMS Run-Time Settings" *in Volume II - the Protocols user guide*
Speed Simulation	Network	"About Network Run-Time Settings" on page 241
Think Time	General	"Configuring the Think Time Settings" on page 231
Timing	Citrix	"Citrix Timing Run-Time Settings" in *Volume II - the Protocols user guide*
Toolkit Options	Web Services Toolkit	"Setting Web Service Toolkit Run-Time Settings" *in Volume II - the Protocols user guide*

Categorical Listing of Run-Time Settings

Category	Run-Time Setting	For more information, see:
Browser	Browser Emulation	"Setting Browser Emulation Properties" in *Volume II - the Protocols user guide*
Citrix	Configuration	"Citrix Configuration Run-Time Settings" in *Volume II - the Protocols user guide*
Citrix	Timing	"Citrix Timing Run-Time Settings" in *Volume II - the Protocols user guide*
General	Additional Attributes	"Configuring Additional Attributes Run-Time Settings" on page 233
General	Log	"Configuring the Log Run-Time Settings" on page 226
General	Miscellaneous	"Configuring Miscellaneous Run-Time Settings" on page 234
General	Pacing	"Configuring Pacing Run-Time Settings (multi-action)" on page 223
General	Run Logic	"Setting Pacing and Run Logic Options (single action)" on page 224
General	Think Time	"Configuring the Think Time Settings" on page 231
Internet Protocol	Content Check	"About Checking Web Page Content" in *Volume II - the Protocols user guide*
Internet Protocol	Download Filters	"Filtering Web Sites" *in Volume II - the Protocols user guide*
Internet Protocol	Preferences	"Setting Internet Preferences" *in Volume II - the Protocols user guide*
Internet Protocol	Proxy	"Setting Proxy Options" in *Volume II - the Protocols user guide*

Java Environment Settings	Classpath	"Setting the Run-Time Classpath Options" in *Volume II - the Protocols user guide*
Java Environment Settings	Java VM	"Specifying the JVM Run-Time Settings" in *Volume II - the Protocols user guide*
JMS	Advanced	"Setting Web Services JMS Run-Time Settings" *in Volume II - the Protocols user guide*
MMS, Multimedia Messaging Service	Server and Protocol	"Configuring MMS Run-Time Settings" *in Volume II - the Protocols user guide*
.NET	.NET Environment	"Configuring .NET Environment Run-Time Settings" *in Volume II - the Protocols user guide*
Network	Speed Simulation	"Configuring Network Run-Time Settings" on page 241
Oracle NCA	Client Emulation	"Configuring Oracle NCA Run-Time Settings" *in Volume II - the Protocols user guide*
RTE	RTE	"Configuring RTE Run-Time Settings" in *Volume II - the Protocols user guide*
SAPGUI	General	"Setting SAPGUI Run-Time Settings" in *Volume II - the Protocols user guide*
WAP	Gateway	"Configuring Gateway Options" in *Volume II - the Protocols user guide*
WAP	Radius	"Configuring Radius Connection Data" in *Volume II - the Protocols user guide*
Web Services Toolkit	Toolkit Options	"Setting Web Service Toolkit Run-Time Settings" *in Volume II - the Protocols user guide*

Recording Options

The recording options let you instruct VuGen how to generate a script and what settings to apply to the Vuser during the recording.

To learn more about a recording option, locate it in one of the tables and refer to the appropriate section as shown in the right column. You can use either of the listings:

- Alphabetical Listing of Recording Options
- Categorical Listing of Recording Options

Alphabetical Listing of Recording Options

Recording Option	Category	For more information, see:
Advanced	GUI Properties	"Recording with Click and Script" on page 397 *in Volume II - the Protocols user guide*
Advanced	HTTP Properties	"Setting Advanced Recording Options" *in Volume II - the Protocols user guide*
Auto Logon	SAPGUI	"SAPGUI Auto Logon Recording Options" in *Volume II - the Protocols user guide*
Classpath	Java Environment Settings	"Setting Classpath Recording Options" in *Volume II - the Protocols user guide*
Code Generation	Citrix	"Code Generation Recording Options" on page 105 in Volume II - the *Protocols user guide*
Code Generation	SAPGUI	"SAPGUI Code Generation Recording Options" in *Volume II - the Protocols user guide*
Code Generation Options	EJB Options	"Setting EJB Recording Options" in *Volume II - the Protocols user guide*

Configuration	Citrix	"Configuration Recording Options" in *Volume II - the Protocols user guide*
Configuration	RTE	"Setting RTE Configuration Options" in *Volume II - the Protocols user guide*
Corba Options	Recording Properties	"CORBA Options" in *Volume II - the Protocols user guide*
Correlation	HTTP Properties	"Setting the Correlation Recording Options" in *Volume II - the Protocols user guide*
Correlation Options	Recording Properties	"Correlation Options" in *Volume II - the Protocols user guide*
Database	Database	"Setting Database Recording Options" in *Volume II - the Protocols user guide*
Debug Options	Recording Properties	"Log Options" in *Volume II - the Protocols user guide*
Filter	COM/DCOM	"Setting the Filter" in *Volume II - the Protocols user guide*
Filters	Microsoft .NET	"Setting a Recording Filter" in *Volume II - the Protocols user guide*
General	Protocols	"Adding and Removing Protocols" on page 77
General	SAPGUI	"SAPGUI General Recording Options" in *Volume II - the Protocols user guide*
Java VM	Java Environment Settings	"Java Virtual Machine (JVM) Recording Options" in *Volume II - the Protocols user guide*
Login	Citrix	"Login Recording Options" in *Volume II - the Protocols user guide*
Options	COM/DCOM	"Setting COM Scripting Options" in *Volume II - the Protocols user guide*
Port Mapping	Network	"Configuring the Port Mappings" on page 109

Recorder	Citrix	"Recorder Recording Options" in *Volume II - the Protocols user guide*
Recorder Options	Recording Properties	"Recorder Options" in *Volume II - the Protocols user guide*
Recording	General	"Selecting a Recording Level" *in Volume II - the Protocols user guide*
Recording	Microsoft .NET	"Setting Microsoft .NET Recording Options" in *Volume II - the Protocols user guide*
RTE	RTE	"Setting the RTE Recording Options" in *Volume II - the Protocols user guide*
Script	General	"Setting Script Generation Preferences" on page 103
Serialization Options	Recording Properties	"Serialization Options" in *Volume II - the Protocols user guide*
Traffic Filters	Traffic Analysis	"Specifying Traffic Information" in *Volume II - the Protocols user guide*
Web Event Configuration	GUI Properties	"Configuring Web Event Recording" in *Volume II - the Protocols user guide*
WinSock	Sockets	"Setting the WinSock Recording Options" in *Volume II - the Protocols user guide*

Categorical Listing of Recording Options

Category	Recording Option	For more information, see:
Citrix	Code Generation	"Code Generation Recording Options" on page 105 in Volume II - the *Protocols user guide*
Citrix	Configuration	"Configuration Recording Options" in *Volume II - the Protocols user guide*
Citrix	Login	"Login Recording Options" in *Volume II - the Protocols user guide*
Citrix	Recorder	"Recorder Recording Options" in *Volume II - the Protocols user guide*
COM/DCOM	Filter	"Setting the Filter" in *Volume II - the Protocols user guide*
COM/DCOM	Options	"Setting COM Scripting Options" in *Volume II - the Protocols user guide*
Database	Database	"Setting Database Recording Options" in *Volume II - the Protocols user guide*
EJB Options	Code Generation Options	"Setting EJB Recording Options" in *Volume II - the Protocols user guide*
General	Protocols	"Adding and Removing Protocols" on page 77
General	Script	"Setting Script Generation Preferences" on page 103
General	Recording	"Selecting a Recording Level" *in Volume II - the Protocols user guide*
GUI Properties	Advanced	"Setting Advanced GUI Properties" in *Volume II - the Protocols user guide*
GUI Properties	Web Event Configuration	"Configuring Web Event Recording" in *Volume II - the Protocols user guide*
HTTP Properties	Advanced	"Setting Advanced Recording Options" *in Volume II - the Protocols user guide*

HTTP Properties	Correlation	"Setting the Correlation Recording Options" in *Volume II - the Protocols user guide*
Java Environment Settings	Classpath	"Setting Classpath Recording Options" in *Volume II - the Protocols user guide*
Java Environment Settings	Java VM	"Java Virtual Machine (JVM) Recording Options" in *Volume II - the Protocols user guide*
Microsoft .NET	Recording	"Setting Microsoft .NET Recording Options" in *Volume II - the Protocols user guide*
Microsoft .NET	Filters	"Setting a Recording Filter" in *Volume II - the Protocols user guide*
Network	Port Mapping	"Configuring the Port Mappings" on page 109
Recording Properties	Corba Options	"CORBA Options" in *Volume II - the Protocols user guide*
Recording Properties	Correlation Options	"Correlation Options" in *Volume II - the Protocols user guide*
Recording Properties	Debug Options	"Log Options" in *Volume II - the Protocols user guide*
Recording Properties	Recorder Options	"Recorder Options" in *Volume II - the Protocols user guide*
Recording Properties	Serialization Options	"Serialization Options" in *Volume II - the Protocols user guide*
RTE	Configuration	"Setting RTE Configuration Options" in *Volume II - the Protocols user guide*
RTE	RTE	"Setting the RTE Recording Options" in *Volume II - the Protocols user guide*
SAPGUI	Auto Logon	"SAPGUI Auto Logon Recording Options" in *Volume II - the Protocols user guide*

SAPGUI	Code Generation	"SAPGUI Code Generation Recording Options" in *Volume II - the Protocols user guide*
SAPGUI	General	"SAPGUI General Recording Options" in *Volume II - the Protocols user guide*
Sockets	WinSock	"Setting the WinSock Recording Options" in *Volume II - the Protocols user guide*

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z